Achieving Target MTTF by Duplicating Reliability-Critical Components in High Performance Computing Systems

Nithin Nakka[‡], Alok Choudhary[†], Gary Grider[§], John Bent[§], James Nunez[§] and Satsangat Khalsa[§]
[‡]Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, USA
nakka@crhc.illinois.edu
[†]Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, USA
choudhar@eecs.northwestern.edu
[§]Los Alamos National Laboratories, Albuquerque, New Mexico, U.S.A.
{ggrider, johnbent, jnunez, satsang}@lanl.gov

Abstract

Mean Time To failure, MTTF, is a commonly accepted metric for reliability. In this paper we present a novel approach to achieve the desired MTTF with minimum redundancy. We analyze the failure behavior of large scale systems using failure logs collected by Los Alamos National Laboratory. We analyze the root cause of failures and present a choice of specific hardware and software components to be made fault-tolerant, through duplication, to achieve target MTTF at minimum expense. Not all components show similar failure behavior in the systems. Our objective, therefore, was to arrive at an ordering of components to be incrementally selected for protection to achieve a target MTTF. We propose a model for MTTF for tolerating failures in a specific component, system-wide, and order components according to the coverage provided. Systems grouped based on hardware configuration showed similar improvements in MTTF when different components in them were targeted for fault-tolerance.

1 Introduction

Computers are being employed increasingly in highly mission- and life-critical and long-running applications. In this scenario, there is a corresponding demand for high reliability and availability of the systems. Since failures are inevitable in a system, the best use of the bad bargain is to employ fault-detection and recovery techniques to meet the requirements. Mean Time to Failure (MTTF) is an important well-accepted measure for the reliability of a system. MTTF is the time elapsed, on an average, between any two failures in the component being studied.

Broadly speaking, two types of applications demand high reliability – (i) those which can be stopped and their state captured at a suitable point and their execution resumed at a later point in time from the captured state, also called the checkpointed state, (ii) those programs that cannot be interrupted and need to execute for a minimum amount of time, till the application (or the mission) is completed. Most long-running scientific applications are examples of applications in the former category. They require efficient mechanisms to take checkpoints of the entire state of the application at a suitable point, and effective techniques to detect failures so as to roll back execution to the checkpointed state. For applications in the latter category, such as systems and software for flight, or spacecraft control, a time for the length of the mission (or mission time) is pre-determined and appropriate fault-tolerance techniques need to be deployed to ensure that the entire system does not fail within the mission time. The mission time of a flight system directly determines the length of its travel and is a highly critical decision point.

The MTTF of a system is an estimate of the time for which the system can be expected to work without any failures. Therefore, for applications that can be checkpointed MTTF could be used to determine the checkpointing interval, within which the application's state must be checkpointed. This would ensure that the checkpoint state itself is not corrupted and hence by rolling back to this state on detecting a failure the application will continue correct execution. For applications of the latter category, the MTTF can be used to determine the mission time, before which the system executes without any failure.

Understanding the failure behavior of a system can greatly benefit the design of fault-tolerance and reliability techniques for that as well as other systems with similar characteristics and thereby increasing their MTTF. Failure and repair logs are a valuable source of field failure information. The extent to which the logs aid in reliable design depends on the granularity at which the logging is performed. System level logs could assist in system-wide techniques such as global synchronous checkpointing etc. However, logging at a finer granularity, like that at the node-level, improves the effectiveness of techniques applied at the node level. An important observation that we make in this paper is that, "All components in a system are not equal" (either by functionality or by failure behavior).

Component-level reliability information also helps in designing and deploying techniques such as

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.311

duplication selectively to the most critical portions of the system. This decreases setup, maintenance and performance costs for the system. Although a technique for achieving a target MTTF has been presented in [3], it does not analyze the root cause of the failures to identify the critical components as in this work. Furthermore, with a framework for selective fault-tolerance in place the techniques could be customized to meet the specific reliability requirements of the application. In this paper we show how component-level selective fault-tolerance can be customized to meet an application's target MTTF.

The key contributions of this work are:

1. Analysis of failures in specific components and their correlation with system configuration.
2. Data-driven estimation of the coverage provided for fault tolerance in components in the system.
3. A methodology for selecting an optimal or near-optimal subset of components to be duplicated to achieve the MTTF requirements of the application.

2 Description of Systems under study and data sets

Los Alamos National Laboratory has collected and published data regarding the failure and usage of 22 of their supercomputing clusters. This data was previously analyzed by Schroeder et. al. [8] from CMU to study the statistics of the data in terms of the root cause of the failures, mean time between failures and the mean time to repair.

The failure data for a single system includes the following fields:

node number: This node numbering is provided by Los Alamos so as to maintain consistency among all systems in terms of node numbering for easier comparison of systems.

install date: Date that the node was installed in the system. Since the systems under study were upgraded during the period of study, this field can vary for nodes within a system

production date: Date, after the installation date, where the node has been tested for initial burn-in effects and known operational failures, so that the node could be used to run production applications.

decommission date: Date that the node was removed from the system.

Problem Started (mm/dd/yy hh:mm): The time of occurrence of the failure event. This is logged by an automated fault detection engine.

Problem Fixed (mm/dd/yy hh:mm): The time at which the failure was repaired and the system restored. This is logged by the administrators and repair staff.

Root cause: Determined by the administrators or maintenance personnel of the systems. This field provides information on the specific component of the node/system that caused the failure. The root causes are broadly classified into *Facilities*, *Hardware*, *Operator Error* (or *Human Error*), *Network Error*, and *Software*. Those failures whose root cause could not be resolved are placed in the *Undetermined* category. In this analysis we consider failures for all systems together as well as for each system at a time. The first approach provides insight on the failure rate of each of the component irrespective of the type of system they are part of. By conditioning this analysis with the system, we can understand how the failure behavior of each of the components changes with the specific type and configuration of the system.

Each of the six broad categories, are further classified into sub-categories or components. The failure analysis traces the root cause of each failure to one of these sub-categories/components.

Table 1 tabulates the failure categories and their corresponding set of sub-categories/components (For brevity we will refer to sub-categories/components as sub-components for the following discussion).

Table 1: Failure Categories and subcomponents

Failure Category	Failing Sub Category or Sub Component
Operator Error	Human Error
Network Error	Network
Undetermined	Security,Unresolvable,Undetermined
Facilities	Environment,Chillers,Power Spike,UPS,Power Outage
Software	Compilers and libraries, Scratch Drive, Security Software, Vizscratch FS, …
Hardware	WACS Logic, SSD Logic, Site Network Interface, KGPSA, SAN Fiber Cable, …

3 Related Work

There has been significant study over the past few decades on analyzing failure logs from large-scale computers to understand the failure behavior of such systems and possibly use the knowledge in improving system design. Plank et. al. [1] derive an optimal checkpointing interval for an application using the failures logs obtained from networks of workstations. They derive the failure rate of the machines using the time between failures observed in the logs. In another study, Nath et. al. [2] study real-world failure traces to recommend design principles that can be used to tolerate correlated failures. Particularly they have used it in recommending data placement strategies for correlated failures.

Previous work in analyzing the failure data set used in this paper aimed at optimizing node level redundancy [3]. In [3] only node-level redundancy was optimized but the root cause of the failures was not considered to identify the critical sub-components of the system as has been done in the present work. The metric used in [3] was erroneously referred to as Mean Time To Failure (MTTF), even though the theory, analysis and results were presented for the metric Mean Time Between Failure (MTBF). It is to be noted that MTBF is not the same as the MTTF. Apart from the MTTF itself, MTBF includes the time required to repair the previous failure. MTTR is defined as the Mean Time To Repair a failure. Therefore, MTBF = MTTF + MTTR. In this paper, we perform the analysis and present the results for the metric MTTF.

Prior work in analyzing failure rates tried to arrive at reasonable curves that fit the failure distribution of the systems [4]-[7]. Schroeder and Gibson [8] have presented the characteristics of the failure data that has been used in this study as well. However, a limitation of these studies is that they do not evaluate how system design choices could be affected by the failure characteristics that they have derived from the data. In this work we specifically attempt to understand the failure behavior and use that knowledge in the design of an optimal component-level fault-tolerance strategy with the aim of increasing the overall availability of the system at the least cost.

There is also interesting research work in understanding the correlations between system parameters and failure rate. Sahoo et. al. [5] show that the workload of the system is closely correlated with its failure rate, whereas Iyer [9] and Castillo [10] bring out the correlation between workload intensity and the failure rate. In our study we study the dependency of failure rate on network configuration, which in turn determines the workload characteristics of the system. For example, a fat tree topology necessitates higher communication and computation bandwidth and load at the higher levels of the tree structure.

Oliner and Stearley [11] have analyzed system logs from five supercomputers and critically evaluate the interpretations of system administrators from patterns observed in the logs. They propose a filtering algorithm to identify alerts from system logs. They also recommend enhancements to the logging procedures so as to include information crucial in identifying alerts from non-alerts.

Lan et. al. [12][13] have proposed a machine-learning based automatic diagnosis and prognosis engine for failures through their analysis of the logs on Blue Gene/L systems deployed at multiple locations. Their goal is to feed the knowledge inferred from the logs to checkpointing and migration tools [14][15] to reduce the overall application completion time.

Oliner et. al. [16] derive failure distributions from multiple supercomputing systems and propose novel job-scheduling algorithms that take into account the occurrence of failures in the system. They evaluated the impact of this on the average bounded slowdown, average response time and system utilization. In our study we have utilized failure and repair time distributions to propose a novel approach to selective fault-tolerance. The metric of evaluation used is the mean time to failure (MTTF) of the system.

Duplication, both at the system- and node- level has been a topic of active and extensive research in the micro-architecture and fault –tolerant computing areas. *Error Detection Using Duplicated Instructions (EDDI)* [20] duplicates original instructions in the program but with different registers and variables. Duplication at the application level increases the code size of the application in memory. More importantly, it reduces the instruction supply bandwidth from the memory to the processor. *Error Detection by Diverse Data and Duplicated Instructions (ED^{4}I)* [21] is a software-implemented hardware fault tolerance technique in which two "different" programs with the same functionality are executed, but with different data sets, and their outputs are compared. The "different" programs are generated by multiplying all variables and constants in the original program by a diversity factor *k*.

In the realm of commercial processors the IBM G5 processor [22] has extra I- and E- units to provide duplicate execution of instructions. To support duplicate execution, the G5 is restricted to a single-issue processor and incurs 35% hardware overhead.

In experimental research, simultaneous multithreading (SMT) [23] and the chip multiprocessor (CMP) architectures have been ideal bases for space and time redundant fault-tolerant designs because of their inherent redundancy. In simultaneously and redundantly threaded (SRT) processor, only instructions whose side effects are visible beyond the boundaries of the processor core are checked [24]-[26]. This was subsequently extended in SRTR to include recovery [19]. Another fault-tolerant architecture is proposed in the DIVA design [17][18]. DIVA comprises an aggressive out-of-order superscalar processor along with a simple in-order checker processor. *Microprocessor-based introspection (MBI)* [27] achieves time redundancy by scheduling the redundant execution of a program during idle cycles in which a long-latency cache miss is being serviced. SRTR [19] and MBI [27] have reported up to 30% performance overhead. These results counter the widely-used belief that full duplication at the processor-level incurs little or no performance overhead.

SLICK [28] is an SRT-based approach to provide partial replication of an application. The goals of this approach are similar to ours. However, unlike this approach we do not rely on a multi-threaded architecture for the replication. Instead, this paper presents modifications to a general superscalar processor to support partial or selective replication of the application.

As for research and production systems employing system-level duplication, the space mission to land on the moon used a TMR enhanced computer system [29]. The TANDEM, now HP, Integrity S2 computer system [30] provided reliability through the concept of full duplication at the hardware level. The AT&T No.5 ESS telecommunications switch [31], [32] uses duplication in its administrative module consisting of the 3B20S processor, an I/O processor, and an automatic message accounting unit, to provide high reliability and availability. The JPL STAR computer [33] system for space applications primarily used hardware subsystem fault-tolerant techniques, such as functional unit redundancy, voting, power-spare switching, coding, and self-checks.

4 Approach

This section describes our approach for analyzing the data and building a model used for selective component-level fault-tolerance. The records for a single component are ordered according to the time of occurrence of the failure, as given by the "Prob Started field". The time elapsed between the repair of one failure and the occurrence of the next failure in this ordered list gives the time to failure for the second failure (In case of the first failure the time to failure is the time from the installation of the component to the failure). Following this procedure the times to failure for each failure for the component are calculated. The average of all these times is used as an estimate for the mean time to failure (MTTF) for the component. Time To Failure (TTF) for a fault *i* is given by:

$$TTF_1 = prob\ started_1 - \ system\ install\ date$$
$$TTF_i = prob\ started_i - \ prob\ fixed_{i-1}$$
$$for\ 2 \le i \le n$$

and therefore,

$$MTTF = \frac{\sum_{i=1}^{n} TTF_i}{n}$$
$$= \frac{prob\ started_1 - \ system\ install\ date + \sum_{i=2}^{n}(prob\ started_i - \ prob\ fixed_{i-1})}{n}$$
$$= \frac{prob\ started_1 - \ system\ install\ date + \sum_{i=2}^{n}\{prob\ started_i - \ (prob\ started_{i-1} + downtime_{i-1})\}}{n}$$
$$= \frac{prob\ started_1 - \ system\ install\ date + \sum_{i=2}^{n}\{(prob\ started_i - prob\ started_{i-1}) - downtime_{i-1}\}}{n}$$
$$= \frac{prob\ started_1 - \ system\ install\ date + prob\ started_n - \ prob\ started_1 - \sum_{i=1}^{n-1} downtime_i}{n}$$
$$= \frac{prob\ started_n - \ system\ install\ date - \sum_{i=1}^{n-1} downtime_i}{n}$$

MTTF is calculated as:

$$MTTF = \frac{Total\ Period\ of\ study - Total\ Downtime\ for\ all\ failures}{Number\ of\ Failures\ observed\ (n)} \qquad \text{Eq. 1}$$

The period of study of a system is its production time, defined elsewhere as the time between its installation and its decommissioning or the end of the observation period, whichever occurs first. Total downtime for all failures is the sum of the downtimes of all failures observed for the system.

The "Downtime" field provides the time required by the administrators to fix the failure and bring the system back to its original state. It can be calculated as the difference between the "Prob Ended" and "Prob Started" fields. This is the repair time for this failure. Averaging this field over all the records for a single component provides an estimate for the mean time to repair (MTTR) for this component. Time To Repair (TTR) for a failure

$$TTR_i = (prob\ fixed)_i - \ (prob\ started)_i.$$

Therefore, Mean Time To Repair is given by

$$MTTR = \frac{\sum_{i=1}^{n} TTR_i}{n}$$

4.1 Introducing component protection

From the previous analysis procedures, the MTTF and the MTTR for a component have been estimated. Now, we introduce a methodology to understand the effect on component failure if we augment it with a spare component. Figure 1 shows the states through which a duplicated component transitions on failure and repair events. When both the original and the spare component are working correctly the system is in state "2". It is assumed that, after a failure is detected in the component, the system has the reconfiguration capability to fail over to the spare component instantaneously. Computation therefore continues uninterruptedly. This state of the component is represented by state “1” in the figure. In the mean time, the original component is repaired. The roles of the spare component and original component are switched. If no other failure occurs in the component, before the original component is repaired, then the original component assumes the role of the spare component, while the computation continues on the spare component. Essentially, the system is brought back to its pristine, fault-free state (State “2” in the figure). However, if the next failure for the component occurs within the mean time to repair for that component, then it is not possible to continue computation on that component. The component reaches state “0”. We declare that protecting this component cannot cover this second failure. There are other possible transitions between these states, shown as dotted lines in Figure 1. They are (i) State “0” to “2”: When both the original and spare components are repaired and the component returns to its normal state. (ii) State “0” to “1”: When one of the failed components (the original or the spare) is repaired and computation continues on this component. (iii) State “2” to “0”: When both components fail at the same time. However, it is to be noted that for the analysis based on the data these transitions need not be considered. There would not be a transition from State “2” to “0” since the data represent failures only in one single component and would not therefore have two simultaneous failures. The purpose of the analysis (aided by the state transition diagram) is to decide whether a particular failure can be covered by protecting this component or not. Once the component reaches State “0” it is declared that the failure cannot be covered by protecting it. Therefore, outward transitions from State “0” (to States “1” and “2”) are not considered.

Based on this analysis, conducted for each component individually, we evaluate all the failures that are covered by providing fault-tolerance to that component. This analysis provides an estimate of the components, which when duplicated, provide the most benefit in terms of improvement in the MTTF of the system. The next part of the study is used to achieve application requirements of MTTF.

Before choosing a component to be duplicated, let τ_{sys}^{-} be the total time the system was in operation and let f_{sys}^{-} be the number of failures. Then the MTTF of the system at this time is given by $MTTF_{sys}^{-} = \frac{\tau_{sys}^{-}}{f_{sys}^{-}}$.

Let f_i be the failures in a component i that are covered by duplicating it and let τ_i be the downtime due to these f_i failures. If component i is duplicated the total time the system is in operation is given by $\tau_{sys}^{+} = \tau_{sys}^{-} + \tau_i$ and the number of failures is $f_{sys}^{+} = f_{sys}^{-} - f_i$. Therefore, the MTTF of the system if component i is duplicated is given by $MTTF_{sys}^{+} = \frac{\tau_{sys}^{+}}{f_{sys}^{+}} = \frac{\tau_{sys}^{-}+\tau_i}{f_{sys}^{-}-f_i}$.

If component i is to be chosen as the next best candidate for duplication in improving the MTTF of the system then:

$$\frac{\tau_{sys}^{-}+\tau_i}{f_{sys}^{-}-f_i} \geq \frac{\tau_{sys}^{-}+\tau_j}{f_{sys}^{-}-f_j} \; \forall \; nodes\; j\; that\; are\; yet\; to\; be\; chosen\; for\; duplication.$$

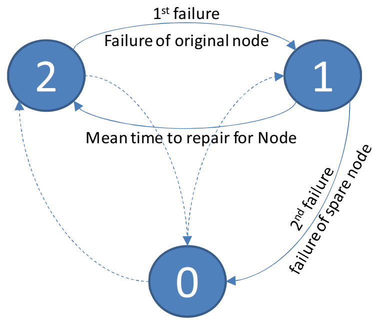


Figure 1: State transition diagram for component failure with single fault-tolerance

We note that the fraction $\frac{\tau_{sys}^{-}+\tau_i}{f_{sys}^{-}-f_i}$ is dependent not only τ_i and f_i but also on $\tau_{sys}^{-}$and f_{sys}^{-}. The choice of the best component i to be chosen cannot be made only by comparing the corresponding τ_i’s and f_i’s. Rather, before making every consecutive choice for the best component, the current $\tau_{sys}^{-}$and f_{sys}^{-} must be noted, and the fraction $\frac{\tau_{sys}^{-}+\tau_i}{f_{sys}^{-}-f_i}$ must be calculated for each component i that has not yet been duplicated. Then the component j that gives the maximum value of $\frac{\tau_{sys}^{-}+\tau_j}{f_{sys}^{-}-f_j}$ is chosen for duplication.

5 Root Cause Analysis

Referring to [8] we see that the 22 systems under study are divided into 8 categories based on the types of CPU and memory and the network configuration. Of these we will limit our analysis to sizeable systems (with more than 500 processors). Thus only Systems of Type E, F and G are considered in this analysis.

5.1 Failures for all systems

Failure data analysis independent of the system brings out the impact of the specific Category and sub-component where the failure occurred. For this reason this specific focuses on the distribution of failures from all systems and their impact. Figure 2 shows the distribution of failures across the six categories. Figure 2 (a) shows the frequency of occurrence of the failures, Figure 2 (b) shows the total downtime caused due to these failures, and Figure 2 (c) shows the average downtime due to each of the six categories. From Figure 2(a) we can see that most of the failures occurred in hardware components of the systems, while human error caused the least number of errors. Figure 2(b) shows that hardware components also had the highest overall impact on the system in terms of their combined contribution to the total downtime. However, when seen on an average per failure, (as shown in Figure 2(c)) a failure occurring in the facilities category had a higher impact (in terms of downtime) than one in any other category.

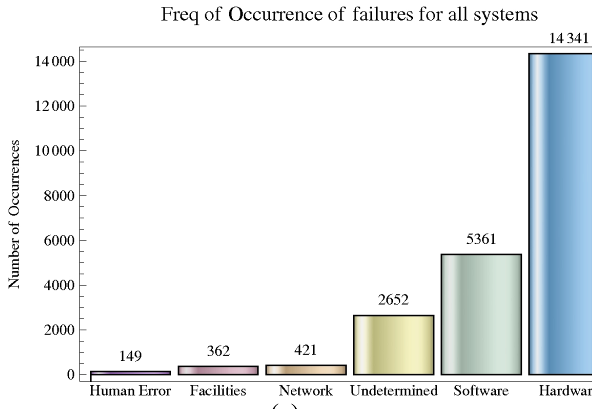
5.2 Failure distribution for all systems within a category

Of the six categories Operator Error, Network Error and Undetermined have only 1 to 3 sub-components. Therefore, we do not consider these in our detailed analysis for failures in sub-components. The data shown consider only failures in Facilities, Software and Hardware categories for all systems put together and for individual systems within a characteristic group. Among the 22 systems, we will focus on the larger systems for the analysis for root cause analysis to understand the most critical components in each system. These large systems are further divided in groups based on their characteristics such as CPU type, Memory Type etc. The groups and the constituent systems are given in Table 2.

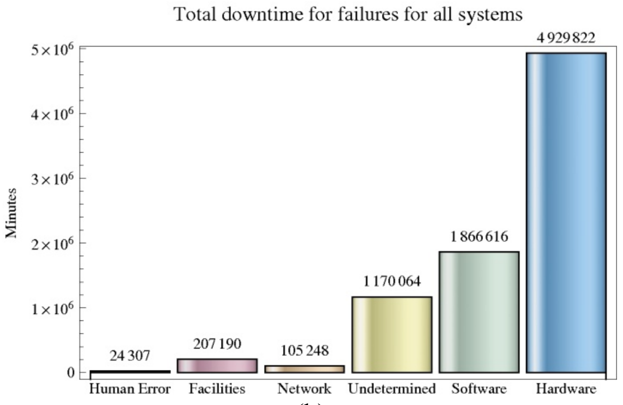
Table 2: Grouping of systems based on configuration

Group	Systems
E	3, 4, 5, 6, 18, 19, 20, 21
F	9, 10, 11, 12, 13, 14
G	16, 2, 23

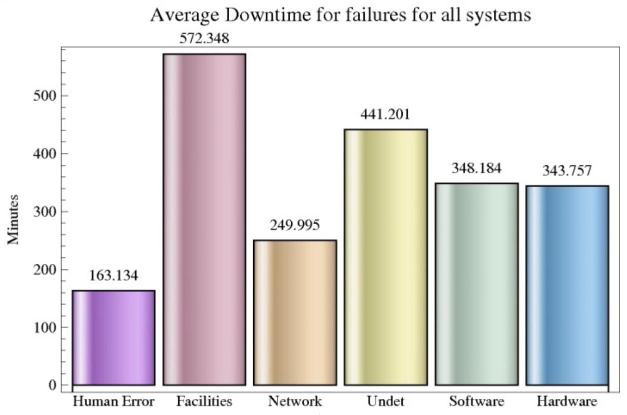
In the previous section it was determined as to which component would provide the highest improvement in MTTF when it is duplicated. We now analyze each system and group failures according to the component in which they occur. Based on this grouping we determine the component, which when protected, provides the best improvement in MTTF. The set of failing components for a system are a subset of those listed in Table 1.



(a)



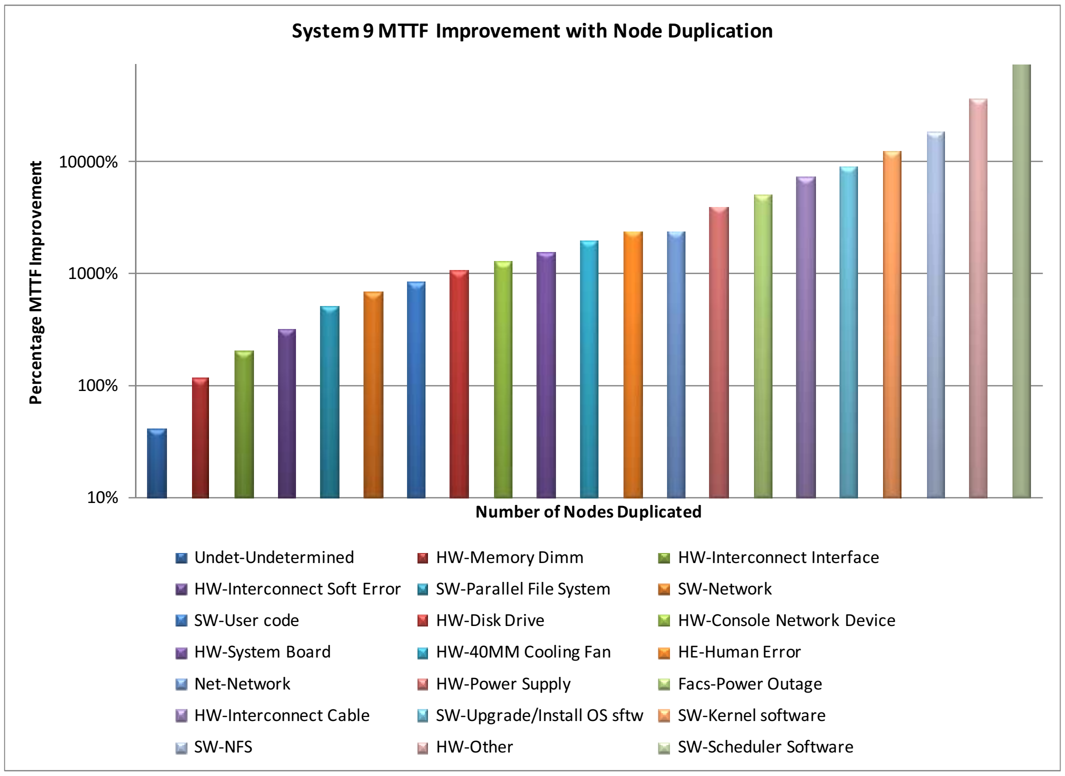
(b)



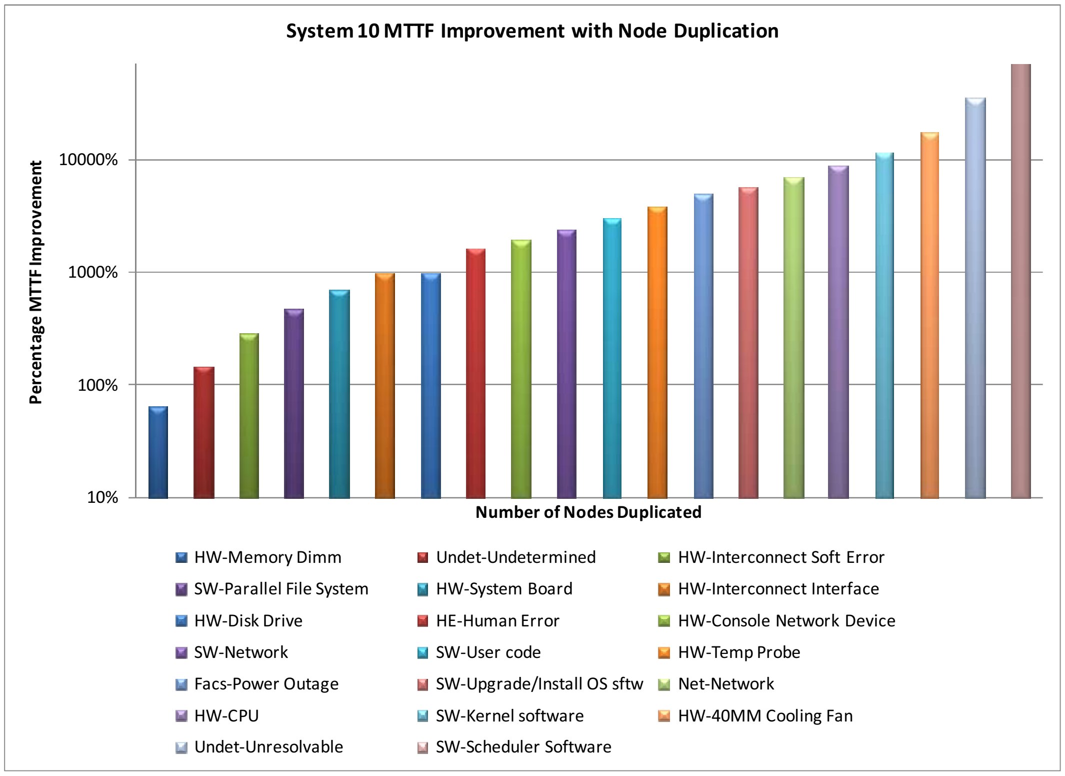
(c)

Figure 2: Failure Distribution for All Systems

The analytical procedure presented in Section 4.1 is used for the components as well. In place of a component, a component throughout the entire system is protected against failures. For example, for failures in CPUs, all CPUs in the system are duplicated to cover any failures. We follow the state diagram shown in Figure 1 to determine the coverage of failures.

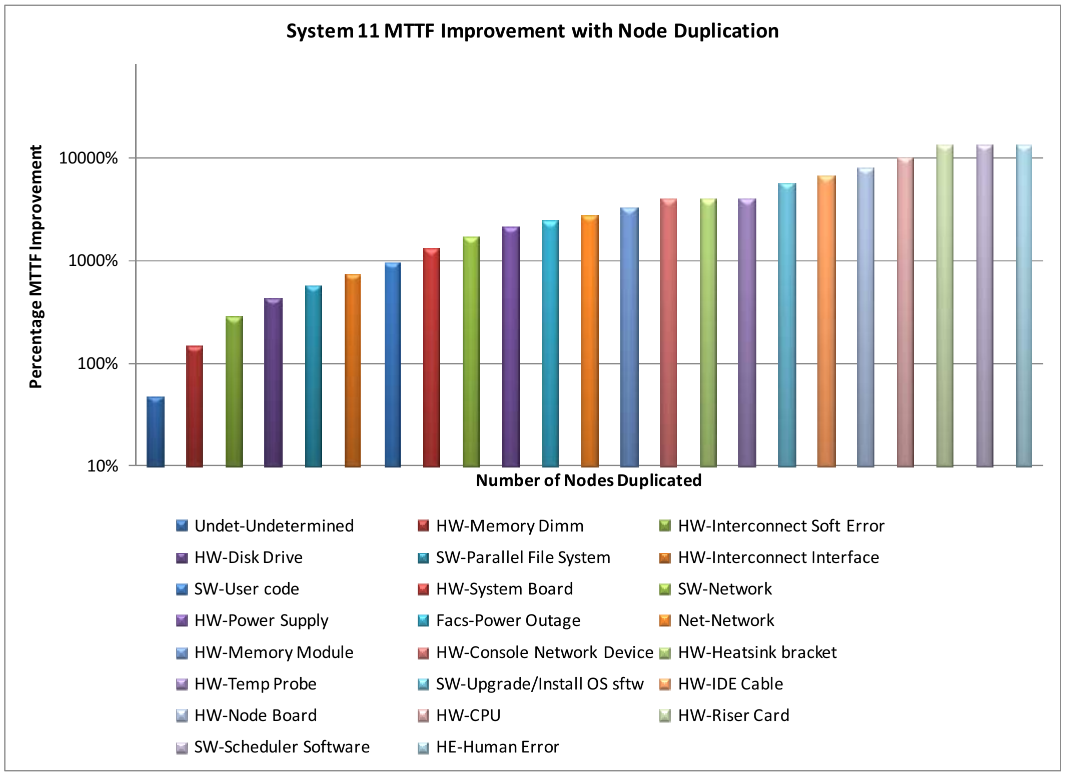


(a)

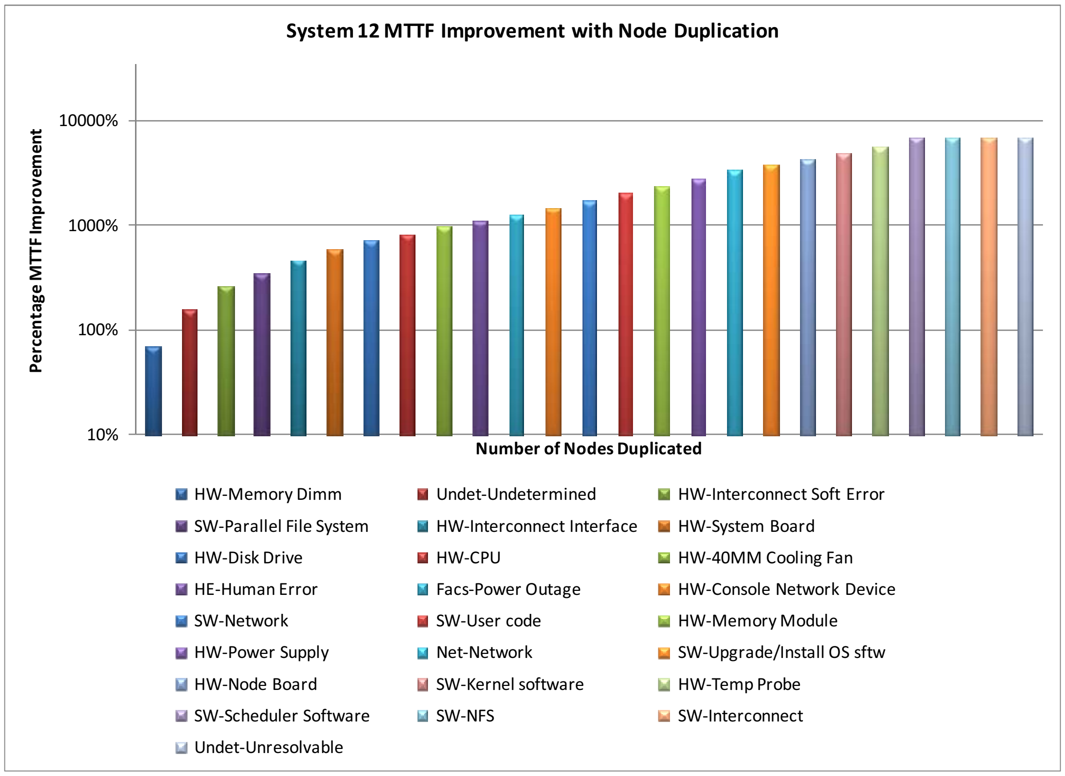


(b)

Figure 3: Improvement in MTTF incrementally covering failures in different components for (a) System 9 (b) System 10 (c) System 11 (d) System 12



(c)



(d)

Figure 4: Improvement in MTTF incrementally covering failures in different components for (a) System 9 (b) System 10 (c) System 11 (d) System 12

As in the analysis shown in Section 4.1 let $\tau_i$be the downtime due to all failures in component i, let f_i be the number of failures occurring in component i, and let τ_{sys}^{-} and f_{sys}^{-} be the total system operation time and failures at the time of choosing the next best component for fault-tolerance. If component i is chosen for protection, then the resultant MTTF is given by: $MTTF_{sys}^{+} = \frac{\tau_{sys}^{-}+\tau_i}{f_{sys}^{-}-f_i}$. The resultant MTTF on protecting all components, one at a time, is calculated. These values are compared and the component providing the highest MTTF is chosen. The order of choosing components for different systems is shown in the following figures.

From Figure 3 we see that systems within a group undergo similar types of failures. The set of failure categories is almost the same for all systems in a group. The curves for the improvement in MTTF of similar systems are also similar showing that the specific components and their order of choice is also more or less similar across the systems in a group. For example, for Systems 9, 10, 11, and 12 HW-Memory Dimm (hardware) is the most critical component, followed by HW-Interconnect (Soft Error/Interface) and so on.

6 Conclusions and future directions

In this paper, we have presented our analysis of the failure behavior of large scale systems using the failure logs collected by LANL on 22 of their computing clusters. We note that not all components show similar failure behavior in the systems. Our objective, therefore, was to arrive at an ordering of components to be incrementally (one by one) selected for duplication so as to achieve a target MTTF for the system after duplicating the least number of components. Using the start times and the down times logged for the failures we derived the time to failures and the mean time for repairs failures on a component. Using these quantities, we arrived at a model for the fault coverage provided by duplicating each component and ordered the components according to MTTF improvement provided by duplicating each component. We analyze the failures grouped by the components in which they occur to understand the critical components and failures types. We observed that systems of similar hardware and software configurations showed similar MTTF improvement when specific components or failure types are targeted for fault tolerance.

The failure data from LANL provides node level failure information even though each node has multiple and different number of processors. Therefore a more fine-grained logging of failures at the processor-level could provide even higher improvement in hardware overheads in achieving higher levels of System-level MTTFs.

A further improvement in the analysis is to include the cost of the component to be protected as a factor in evaluating the necessity to duplicate it. This would determine the most ideal choice of components for fault-tolerance to achieve a particular target given a certain reliability budget for the system.

References

[1] J. S. Plank and W. R. Elwasif. Experimental assessment of workstation failures and their impact on checkpointing systems. In Proceedings of *FTCS-98*.

[2] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan. Subtleties in tolerating correlated failures. In Proceedings of NSDI'06, 2006.

[3] N. Nakka, A. Choudhary, "Failure data-driven selective node-level duplication to improve MTTF in High Performance Computing Systems", In Proceedings of HPCS 2009, June 2009, Kingston, Ontario, CA.

[4] T. Heath, R. P.Martin, and T. D. Nguyen. Improving cluster availability using workstation validation. In Proceedings of ACM SIGMETRICS, 2002.

[5] R. K. Sahoo, R. K., A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In Proceedings of Dependable Systems and Networks, June 2004.

[6] D. Tang, R. K. Iyer, and S. S. Subramani. Failure analysis and modelling of a VAX cluster system. In Fault Tolerant Computing Systems, 1990.

[7] J. Xu, Z. Kalbarczyk, and R. K. Iyer. Networked Windows NT system field failure data analysis. In Proc. of the PRDC, 1999.

[8] B. Schroeder and G. Gibson. A large-scale study of failures in high-performance-computing systems. In Proceedings of the DSN, Philadelphia, PA, June 2006.

[9] R. K. Iyer, D. J. Rossetti, and M. C. Hsueh. Measurement and modeling of computer reliability as affected by system activity. ACM Transactions on Computing Systems, Vol. 4, No. 3, 1986.

[10] X. Castillo and D. Siewiorek. Workload, performance, and reliability of digital computing systems. In the 11th FTCS, 1981.

[11] Adam J. Oliner, Jon Stearley: What Supercomputers Say: A Study of Five System Logs. In Proceedings of the DSN, Edinburgh, UK, June 2007, pp. 575-584.

[12] Z. Lan, Y. Li, P. Gujrati, Z. Zheng, R. Thakur, and J. White, "A Fault Diagnosis and Prognosis Service for TeraGrid Clusters", In *Proceedings of TeraGrid'07* , 2007.

[13] P. Gujrati, Y. Li, Z. Lan, R. Thakur, and J. White,"Exploring Meta-learning to Improve Failure Prediction in Supercomputing Clusters", In *Proceedings of ICPP*, 2007.

[14] Y. Li and Z. Lan, "Using Adaptive Fault Tolerance to Improve Application Robustness on the TeraGrid", In *Proceedings of TeraGrid'07* , 2007.

[15] Z. Lan and Y. Li, "Adaptive Fault Management of Parallel Applications for High Performance Computing", IEEE Transactions on Computers , 57(12), pp. 1647-1660.

[16] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam. Fault-aware job scheduling for

Bluegene/L systems. In Proceedings of the 18th IPDPS, 2004.

[17] C. Weaver and T. Austin. "A fault tolerant approach to microprocessor design," in Proceedings of DSN, July 2001, pp. 411-420.

[18] T. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in Proceedings of the Thirty-Second International Symposium on Microarchitecture, November 1999, pp. 196-207.

[19] T. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient fault recovery using simultaneous multithreading," in Proceedings of the Twenty-Ninth Annual International Symposium on Computer Architecture, May 2002, pp. 87-98.

[20] N. Oh, P.P. Shirvani, and E.J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," IEEE Transactions on Reliability, vol. 51(1), pp. 63-75, Mar. 2002.

[21] N. Oh, S. Mitra, and E.J. McCluskey, "ED4I: Error Detection by Diverse Data and Duplicated Instructions," IEEE Transactions on Computers, vol. 51(2), pp. 180-199, Feb. 2002.

[22] T. Slegel, et al. "IBM's S/390 G5 microprocessor design," IEEE Micro, vol. 19(2), pp. 12–23, 1999.

[23] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip performance," in Proceedings of the Twenty-Second International Symposium on Computer Architecture, June 1995, pp. 392-403.

[24] E. Rotenberg, "AR-SMT: A microarchitectural approach to fault tolerance in microprocessors," in Proceedings of the Twenty-Ninth International Symposium on Fault-Tolerant Computing Systems, June 1999, pp. 84-91.

[25] K. Sundaramoorthy, Z. Purser, and E. Rotenberg, "Slipstream processors: Improving both performance and fault tolerance," In Proceedings of the Thirty-Third International Symposium on Microarchitecture, December 2000, pp. 269-280.

[26] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in Proceedings of the Twenty-Seventh International Symposium on Computer Architecture, June 2000, pp. 25-36.

[27] M. A. Qureshi, O. Mutlu, and Y. N. Patt, "Microarchitecture-based introspection: A technique for transient-fault tolerance in microprocessors," In Proceedings of International Conference on Dependable Systems and Networks, June 2005, pp. 434-443.

[28] A. Parashar, A. Sivasubramaniam, S. Gurumurthi. "SlicK: slice-based locality exploitation for efficient redundant multithreading," in Proceedings of the 12th Intl., conference on ASPLOS, 2006.

[29] A.E. Cooper and W.T. Chow, "Development of on-board space computer systems," IBM Journal of Research and Development, vol. 20, no. 1, pp. 5-19, January 1976.

[30] D. Jewett, "Integrity S2: A fault-tolerant Unix platform," Digest of Papers Fault-Tolerant Computing: The Twenty-First International Symposium, Montreal, Canada, pp. 512 - 519, June 25-27, 1991.

[31] "AT&T 5ESS™ from top to bottom," http://www.morehouse.org/hin /ess/ess05.htm.

[32] AT&T Technical Staff. "The 5ESS switching system," The AT&T Technical Journal, Vol. 64(6), Part 2, July-August 1985.

[33] A. Avizienis, "Arithmetic error codes: Cost and effectiveness studies for Application in digital system design," IEEE Transactions on Computers, vol. 20, no. 11, pp. 1332-1331, November 1971.