2010 39th International Conference on Parallel Processing

Optimizing HPC Fault-Tolerant Environment: An Analytical Approach

Hui Jin*, Yong Chen § †, Huaiyu Zhu*, Xian-He Sun*

*Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois, USA
{hjin6, hzhu12,sun}@iit.edu

†Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
cheny@ornl.gov

***Abstract*—The increasingly large ensemble size of modern High-Performance Computing (HPC) systems has drastically increased the possibility of failures. Performance under failures and its optimization become timely important issues facing the HPC community. In this study, we propose an analytical model to predict the application performance. The model characterizes the impact of coordinated checkpointing and system failures on application performance, considering all the factors including workload, the number of nodes, failure arrival rate, recovery cost, and checkpointing interval and overhead. Based on the model, we gauge three parameters, the number of compute nodes, checkpointing interval, and the number of spare nodes to conduct a comprehensive study of performance optimization under failures. Performance scalability under failures is also studied to explore the performance improvement space for different parameters. Experimental results from both synthetic and actual system failure logs confirm that the proposed model and optimization methodologies are effective and feasible.**

***Keywords*-Fault Tolerance; Checkpointing; High-Performance Computing; Performance Optimization; Scalability**

I. Introduction

High-Performance Computing (HPC) has crossed the Petaflop (10^{15} FLOPS) mark and is moving forward to reach the Exaflop (10^{18} FLOPS) range. Teraflop computers (10^{12} FLOPS) are already widely deployed. Such ultra-scale computing power comes with increased system complexity. Modern high-performance computers are often equipped with hundreds of thousands of CPUs, tera-bytes of main memory, and peta-bytes of storage. The top ten supercomputers of top 500 list in November 2009 are equipped with cores ranging from 41616 to 294912 [21]. The large ensemble size of modern parallel systems significantly increases the possibility of system failures. The *Mean-Time-Between-Failures* (MTBF) of a large-scale cluster could drop to hours [19]. The frequent interruption due to failures will severely degrade the application performance.

Checkpointing/Restart (C/R) is a widely used mechanism for fault-tolerant computing. Modeling the application performance with the presence of failures and the C/R system is a key to achieve optimal performance. Traditionally, the application performance is characterized by workload, the number of processes, and the computing power available in a failure-free environment. Failures and checkpointings introduce several more performance factors, including failure arrival rate, failure recovery cost, and checkpointing interval and overhead. The problem is further complicated by the fact that these factors may be correlated. For example, increasing the number of processes will also increase failure arrival rate and checkpointing overhead.

While some parameters have monotonic effect on the application performance, there are several tunable parameters that complicate the performance optimization, such as the number of processes deployed for an application and the checkpointing interval. The selection of larger number of processes can benefit the application by decreasing the workload on each node, but may introduce higher possibility of failures and checkpointing overhead for coordinated checkpointing. The selection of checkpointing interval also plays a critical role in performance. A lazy checkpointing mechanism reduces the checkpointing overhead but penalizes the performance with more work loss in case of failures. On the other hand, a frequent checkpointing involves more overhead with less work loss.

The usage of spare nodes as the replacement is a promising approach to reducing failure recovery cost [18]. However, identifying an appropriate spare node allocation is not an easy task. Excessive spare node allocation is a waste of computing power. On the other hand, the performance can be severely degraded if there are not enough spare nodes and the application has to wait for the failures to be manually repaired [17].

All of the issues discussed above make performance optimization under failures a challenging task. In this study, we present a comprehensive model to characterize the application performance under failures and introduce optimization strategies to utilize checkpointing and parallel processing for better performance. The contribution of this research is threefold:

- First, we propose a queueing model to predict application performance with the consideration of workload, the number of processes, failure arrival rate, recovery cost, and checkpointing interval and overhead.
- Second, we introduce numerical approaches to deriving

§ This work was performed while he was at Illinois Institute of Technology.

0190-3918/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPP.2010.80

the optimal number of processes, checkpointing interval and number of spare nodes for performance optimization.
- Third, we study the scalability with different system parameters, analyze performance improvement space, and provide directions for developing scalable computing environments under failures and checkpointing.

The rest of this paper is organized as follows. We first present the performance model under failures and coordinated checkpointing in section II. Section III introduces optimization methodologies for the tunable parameters. Scalability analysis and its implications are presented in section IV, followed by an experimental study presented in section V to verify the model and the optimization methodologies. Section VI reviews and compares the related work. Finally, section VII concludes this study and discusses future work.

II. Performance under Failures and Coordinated Checkpointing

A. Assumptions

A computation intensive application with *sequential workload* of W in terms of running time (e.g. hours) is assumed to be running on a homogeneous large-scale cluster system. Sequential workload refers to the failure-free execution time of the application with only one process. The application is *completely parallelizable* with a processes, which means that all the components of the application can be parallelized. The workload is evenly distributed to a processes such that each process bears a workload of $w = W/a$. Each process is mapped onto one compute node for execution. Since the processes and compute nodes are one-to-one mapped, we use these two terms interchangeably throughout this paper.

We assume the inter-arrival times of failures for each node are independent and identically distributed (*iid*) as exponential random variables with parameter λ_f, which is the inverse of MTBF. The entire application will be halted if any process fails during the execution so the failure arrivals of the parallel system with a nodes is also exponentially distributed with parameter $\lambda = a\lambda_f$. Though exponential distribution may not be the best fit for failure arrivals with the consideration of correlated failures [22] [19], it is still widely assumed by the important works in this area [25] [18] [4] [24]. One possible reason is that it is too complex, if not impossible, to propose a practical analytical model with other distributions such as *weibull*. Furthermore, exponential distribution will be more suitable for failure arrivals with the assistance of data pre-processing techniques that remove the redundant/correlated failures [28]. In sub-section V-A, we demonstrate that in real system in production, although the failure arrival is not perfectly exponential distributed, our assumption and model can still lead to a high accuracy in matching real scenario.

Applications have two options to handle the interrupts caused by failures. The first option, *repair-based failure recovery*, suspends the application till the failures are physically fixed. As an alternative, the application may also select one or more pre-allocated spare nodes to replace the failed compute nodes and continue the execution, which is termed as *replacement-based failure recovery*. The essence of the second option is to decrease the failure recovery cost. The physical repair of failures by system administrators may take several hours [19], and we can reduce the recovery cost to several minutes, or even seconds, by using the spare nodes [23]. Regardless of the failure handling mechanism, we assume that the recovery time of each failure follows

Table I: Nomenclature

Symbol	Default Value	Description.
W	524,288 Hours	Sequential workload of a parallel application in terms of hours.
a	N/A	Number of processes of an application (Number of compute nodes involved).
w	W/a	Workload on each node in terms of hours.
λ_f	$\frac{1}{8192Hours}$	Failure arrival rate of each node $\frac{1}{MTBF}$.
λ	$a \times \lambda_f$	Failure arrival rate of the system with a nodes.
μ_f	0.1 Hour	Mean failure recovery cost from the perspective of application level.
σ_f	μ_f	Standard deviation of recovery cost in terms of hours.
θ_f	1	Coefficient of variation of failure recovery cost, $\frac{\sigma_f}{\mu_f}$.
τ	0.5 Hour	Checkpointing interval in terms of hours.
δ	$\delta = p + q \times a$	Checkpointing overhead in terms of hours.
p	0.05 Hour	I/O overhead of image writing in terms of hours.
q	0.0006 Hour	Message passing overhead in terms of hours.
γ	$\gamma = \tau + \delta$	Failure-free period needed by a successful checkpointing segment.
T	N/A	Random variable to finish a segment.
X_i/X	N/A	Random variable that reflects rework cost in terms of hours.
Y_i/Y	N/A	Random variable that reflects system down time due to one or more failures.
S	N/A	Random variable that reflects the number of interrupts to finish one segment.
$U_X(S)$	N/A	$\sum_{i=1}^{S} X_i$.
$U_Y(S)$	N/A	$\sum_{i=1}^{S} Y_i$.
$\mathcal{T}$	N/A	Random variable to finish a parallel application in terms of hours.
m	$\lfloor w/\tau \rfloor$	Number of segments with length of $\gamma = \tau + \delta$ required to finish a workload of w on each node.
α	$w \pmod{\tau}$	Length of the last segment to finish the application.
φ	$\frac{1}{2Hours}$	Failure repair rate, inverse of physical failure repair time.
ρ	λ/φ	System-level failure intensity. Must be less than 1 to guarantee the stability of the system.
σ	$1/\varphi$	Standard deviation of physical failure repair cost.
a^s_{opt}	$\frac{0.99\varphi}{\lambda_f}$	System-level factor to determine the optimal number of compute nodes.
a^a_{opt}	N/A	Application-level factor to determine the optimal number of processes.
a_{opt}	$min\{a^s_{opt}, a^a_{opt}\}$	Optimal number of processes that leads to the minimum application execution time while keeping the stability of the system.
τ_{first}	N/A	First order estimation of the optimal checkpointing interval.
τ_{opt}	N/A	Optimal checkpointing interval in terms of hours.
b	N/A	Optimal number of spare nodes.
n	N/A	Random variable of the number of failures in a system with $a + b$ nodes.

a general distribution with mean μ_f and standard deviation σ_f. Replacement-based failure recovery differs from repair-based recovery in the sense that it has lower values of μ_f. The coefficient of variation of failure recovery captures the normalized dispersion of failure recovery time and is denoted as $\theta_f = \sigma_f/\mu_f$. Failures may overlap with each other, which means that a failure may occur during the recovery of a previous failure. In such case, the failures are handled on *First-Come-First-Serve* (FCFS) basis. The latter failure will be queued and recovered till all the previous failures are fixed. A parallel application with a processes can be considered as an M/G/1 queueing system when we treat a failure as an event (customer) arrived in the queue.

Coordinated checkpointing is taken at fixed interval of τ. Similar as [14], we model coordinated checkpointing overhead as $\delta = p+q\times a$, where p is the I/O overhead and $q\times a$ reflects coordination cost. I/O overhead of p does not scale with a under the assumption of completely parallelizable applications. Coordination cost is linearly proportional with the number of processes based on the analysis of Chandy-Lamport algorithm [3], which is the basis of all the current coordinated checkpointing implementations [20] [2] [9]. We assume synchronous checkpointing in this study and do not distinguish between checkpointing overhead and latency. We define *rework* as the work done between the last checkpoint and the arrival of the failure, which has to be redone after the system recovers from the last checkpoint.

Table I lists the symbols, their default values if available, and the descriptions that are frequently used throughout this paper. The default values are selected based on the failure trace from systems in production or related work [19] [14] [15].

B. Performance of Single Checkpointing Segment

The terminology of *checkpointing segment* (or *segment* for simplicity) represents a period of time between two consecutive checkpoints. A segment begins with a stable state of last checkpoint and ends when a new checkpointing is finished. A continuous failure-free time period with length of $\gamma = \tau + \delta$ is the condition to terminate a segment. If a failure happens at time $t < \gamma$, an interrupt occurs and we have to resume the segment after the failure recovery, which initializes another attempt to finish the segment. Such attempts to a successful segment execution iterate till the termination condition is met. The success of a segment execution means that the workload of τ is saved to the stable storage by checkpointing.

Let random variable T denote the time to finish a segment of length γ, we have

$$\begin{aligned} T &= X_1 + X_2 + ... + X_S + Y_1 + Y_2 + ... + Y_S + \gamma \\ &= \sum_{i=1}^{S} X_i + \sum_{i=1}^{S} Y_i + \gamma \\ &= U_X(S) + U_Y(S) + \gamma \end{aligned} \tag{1}$$

where $X_i (1 \leqslant i \leqslant S)$ represents the rework cost and $Y_i (1 \leqslant i \leqslant S)$ is the system down time due to one or more failures. S is the random variable denoting the number of interrupts occurred over the segment. We use X and Y to replace X_i and Y_i throughout this paper when the context is clear. $U_X(S)$ and $U_Y(S)$ represent the summation of X_1 to X_S, Y_1 to Y_S, respectively. Figure 1 illustrates a segment and its composition.

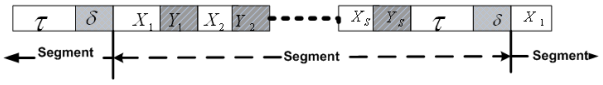


Figure 1: Checkpointing Segment

X falls within the range of $0 < t < \gamma$ and its probability of equalling to t is the conditional probability that a failure happens at time t, given that the inter-arrival of failures is less than γ. So we have the probability density function of X as $f_X(X = t) = \frac{\lambda e^{-\lambda t}}{1-e^{-\gamma\lambda}}$. The mean and variance of X can be expressed as:

$$E(X) = \int_{t=0}^{\gamma} t f_X(X = t) = \frac{1}{\lambda} + \frac{\gamma}{1 - e^{\gamma\lambda}} \tag{2}$$

$$V(X) = E(X^2) - E^2(X) = \frac{1}{\lambda^2} - \frac{e^{\gamma\lambda}\gamma^2}{(1 - e^{\gamma\lambda})^2} \tag{3}$$

Under the assumption of the M/G/1 failure processing, we can get the mean and variance of Y using existing results from queueing theory [11] as:

$$E(Y) = \frac{\mu_f}{1 - \lambda\mu_f} \tag{4}$$

$$V(Y) = E(Y^2) - E^2(Y) = \frac{\sigma_f^2 + \lambda\mu_f^3}{(1 - \lambda\mu_f)^3} \tag{5}$$

S is the number of failed attempts before a successful one, so we have the probability function of random variable S as $P(S = s) = (1-e^{-\gamma\lambda})^s e^{-\gamma\lambda}$. The first term of $(1-e^{-\gamma\lambda})^s$ is the probability of first failed s attempts due to interrupts. The second term of $e^{-\gamma\lambda}$ represents the probability of continuous failure-free segment execution. We can derive the mean and variance of S as:

$$E(S) = \frac{1 - e^{-\gamma\lambda}}{e^{-\gamma\lambda}} = e^{\gamma\lambda} - 1 \tag{6}$$

$$V(S) = \frac{1 - e^{-\gamma\lambda}}{e^{-2\gamma\lambda}} = e^{\gamma\lambda}(e^{\gamma\lambda} - 1) \tag{7}$$

Put the formulas together, we have the mean and variance of T as:

$$\begin{aligned} E(T) &= E(E(T|S)) \\ &= E\left(\gamma + X_1 + X_2 + ... + X_S + Y_1 + Y_2 + ... + Y_S|S\right) \\ &= E(\gamma + SE(X) + SE(Y)) \\ &= (e^{\gamma\lambda} - 1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda}) \end{aligned} \tag{8}$$

$$\begin{aligned} V(T) &= E(V(T|S)) + V(E(T|S)) \\ &= E\left(S\left(V\left(X\right) + V\left(Y\right)\right)\right) + V(S)(E(X) + E(Y))^2 \end{aligned} \tag{9}$$

C. Performance of Parallel Applications

Each segment of $\gamma = \tau + \delta$ discussed in sub-section II-B finishes a useful amount of work τ, which has been successfully checkpointed to the stable storage. We need $m = \lfloor w/\tau \rfloor$ segments of γ and one single segment of $\alpha = w \pmod{\tau}$ to finish the parallel application.

Let $\mathcal{T}$ be the time to finish a parallel application under failures, we have

$$\begin{aligned} E(\mathcal{T}) &= mE(T) + E(T') \\ &= \lfloor w/\tau \rfloor (e^{\gamma\lambda} - 1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda}) \\ &+ (e^{\alpha\lambda} - 1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda}) \\ &\approx (w/\tau)(e^{\gamma\lambda} - 1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda}) \\ &= (\frac{W}{a\tau})(e^{(\tau+p+qa)a\lambda_f} - 1)(\frac{1}{a\lambda_f} + \frac{\mu_f}{1-\mu_f a\lambda_f}) \end{aligned} \tag{10}$$

The first term of $mE(T)$ reflects the mean time of successful m checkpointing segments. The second term of $E(T')$ is the mean time of the rest workload to finish the application. We neglect the second term due to the fact that it is trivial for application with large workload and reasonably short checkpointing interval. The variance of $\mathcal{T}$ is also derivable by $V(\mathcal{T}) = mV(T) + V(T')$.

This theoretical analysis shows that the application execution time and its variation are actually functions of the number of compute nodes a, failure arrival rate of each node λ_f, failure recovery mean μ_f and stand deviation σ_f,checkpointing interval τ and its overhead $\delta = p + q \times a$.

III. Performance Optimization

In section II we have presented a model to characterize and predict the performance of parallel applications under failures, which considers a wide range of parameters. While the application execution time is monotonically impacted by the parameters of failure arrival rate, recovery time and checkpointing overhead, there are several tunable parameters to decide the optimal performance. The rest of this section discusses the optimal selection of the tunable parameters.

A. Optimal Number of Processes

We denote a_{opt} as the optimal number of processes that leads to the minimum application execution time while keeping the *stability* of the system [11]. A system is referred as unstable if the number of failed nodes keeps increasing, otherwise the system is stable.

The number of processes for a parallel application is determined by two factors: *System-Level Factor* a^s_{opt} and *Application-Level Factor* a^a_{opt}.

1) System-Level Factor: From the system's perspective, every failed node must be physically repaired in order to keep the sustainability of the system. We term φ as the failure repair rate, which is the inverse of mean physical failure repair time. Note that the value of φ is determined by the physical failure repair time regardless of the failure handling mechanisms taken by the application. The replacement-based failure recovery helps reduce the failure recovery cost for the corresponding application. However, from the system's view, a failed node eventually requires physical failure repair such that it can be available for future use.

We define $\rho = \lambda/\varphi = a\lambda_f/\varphi$ as the *failure intensity* of the system. Based on queueing theory, the value of ρ must be kept less than 1 to ensure the stability of the system. By setting $\rho = 0.99$ for the maximum value of failure intensity, we have $a^s_{opt} = \frac{0.99\varphi}{\lambda_f}$.

2) Application-Level Factor: While system-level factor a^s_{opt} guarantees the stability of the system, a^a_{opt} is the optimal number of processes that leads to the minimum application execution time. We have $a_{opt} = min\{a^s_{opt}, a^a_{opt}\}$ to guarantee both the system stability and the minimum application execution time.

Considering the expected application execution time of formula (10) as a function of the number of processes a, a^a_{opt} is the value that satisfies the differentiation function of $\frac{\partial E(\mathcal{T})}{\partial a} = 0$, which is a transcendental equation. There is no analytical solution to such equation. We use *Newton's Method* [1] to find the optimal number of processes a numerically.

The pseudocode to find the optimal number of processes is demonstrated in Algorithm 1. We first calculate the a^s_{opt}, which is also set as the starting point of a^a_{opt} for Newton's method. After the iteration terminates, we select $min\{a^s_{opt}, a^a_{opt}\}$ as the optimal number of processes.

Algorithm 1 Find the Optimal Number of Processes

Definition: $g(a) \equiv \frac{\partial E(\mathcal{T})}{\partial a}$. ε is a positive real number sufficiently close to zero.
Objective: Find the optimal number of processes

```
a^s_opt ← 0.99φ / λ_f
a^a_opt ← a^s_opt
while |g(a^a_opt)| > ε do
    a^a_opt ← a^a_opt − g(a^a_opt) / g'(a^a_opt)
end while
a_opt ← min{a^s_opt, a^a_opt}
```

B. Optimal Checkpointing Interval

In this sub-section we discuss the optimal checkpointing interval and its derivation. We first present a first order estimation of the optimal checkpointing interval, followed by a numerical methodology for a higher order estimation.

1) First Order Approximation: The optimal checkpointing interval τ can be derived by solving equation $\frac{\partial E(\mathcal{T})}{\partial \tau} = 0$. For the first order estimation, we have the following conditions hold for relatively reliable system:

- $\lim_{\gamma\lambda \to 0} E(X) = \lim_{\gamma\lambda \to 0}(\frac{1}{\lambda} + \frac{\gamma}{1-e^{\gamma\lambda}}) = \frac{\gamma}{2}$.
- By applying *Taylor Expansion* to $e^{\gamma\lambda}$ and neglecting higher order terms, we have $e^{\gamma\lambda} = 1 + \gamma\lambda + \frac{(\gamma\lambda)^2}{2} + ... \approx 1 + \gamma\lambda$.

Putting all the elements together, we have the simplified version of the mean application execution time as,

$$\begin{aligned} E(\mathcal{T}) &= \frac{w}{\tau}E(T) = \frac{w}{\tau}(\gamma + SE(X) + SE(Y)) \\ &= \frac{w}{\tau}(\tau + \delta + (\tau+\delta)\lambda(\frac{\tau+\delta}{2} + \frac{\mu_f}{1-\lambda\mu_f})) \end{aligned} \tag{11}$$

The differentiation equation can be obtained as $\frac{\partial E(\mathcal{T})}{\partial \tau} = w(\frac{\lambda}{2} - \frac{\delta}{\tau^2}(1 + \delta\lambda/2 + \frac{\mu_f}{1-\lambda\mu_f}\lambda)) = 0$. We obtain the first order approximation of checkpointing interval as

$$\begin{aligned} \tau_{first} &= \sqrt{2\delta(\frac{1}{\lambda} + \delta/2 + \frac{\mu_f}{1-\lambda\mu_f})} \\ &\approx \sqrt{2\delta(\frac{1}{\lambda} + \frac{\mu_f}{1-\lambda\mu_f})} \end{aligned} \tag{12}$$

This result is consistent with the conclusion from [4]. It is also identical to the conclusion of [25] if the failure recovery cost is considered to be zero.

2) Higher Order Approximation: The assumption that $\gamma\lambda$ is negligible made by first order estimation will not hold for large-scale systems with a large value of a, which will increase both $\gamma = \tau + p + q \times a$ and $\lambda = a \times \lambda_f$. Next we introduce a numerical solution to finding a more precise value of optimal checkpointing interval τ.

Again, equation $\frac{\partial E(\mathcal{T})}{\partial \tau} = 0$ is used to derive the optimal value of τ when considering $E(\mathcal{T})$ as a function of τ. From formula (10), we have

$$\begin{aligned} \frac{\partial E(\mathcal{T})}{\partial \tau} &= \frac{\partial (w/\tau)(e^{\gamma\lambda}-1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda})}{\partial \tau} \\ &= w(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda})(\frac{1}{\tau^2})(1 - e^{\delta\lambda}e^{\tau\lambda}(1-\tau\lambda)). \end{aligned} \tag{13}$$

By solving equation $1 - e^{\delta\lambda}e^{\tau\lambda}(1-\tau\lambda) = 0$, we obtain the optimal checkpointing interval. Numerical method similar as Algorithm 1 can also be adopted. In order to converge faster, we suggest to use first order estimation as the starting point of the iteration.

C. Combined Optimization

In sub-sections III-A and III-B we have presented methodologies to derive the optimal number of processes and checkpointing interval individually. Some systems may grant more flexibility such that the user can specify both the number of processes and the checkpointing interval when submitting the job. In this case, the combined optimization from both sides is desired. This question is to seek a combination of (a_{opt}, τ_{opt}) to minimize the application execution time while the other parameters, i.e., λ_f, μ_f, p and q are known.

A brute force solution to this problem is to iterate all possible combinations and find the one with minimum execution time, which is costly, if not impossible for large-scale system with millions of components.

This problem can be formalized as follows, $A_n = \begin{pmatrix} a_n \\ \tau_n \end{pmatrix}$, $f(a) \equiv \frac{\partial E(\mathcal{T})}{\partial a}$, $g(\tau) \equiv \frac{\partial E(\mathcal{T})}{\partial \tau}$ and $F(A_n) = \begin{pmatrix} f(a_n) \\ g(\tau_n) \end{pmatrix}$. If $A_{opt} = \begin{pmatrix} a_{opt} \\ \tau_{opt} \end{pmatrix}$ is the optimal result, it satisfies $F(A_{opt}) = \begin{pmatrix} f(a_{opt}) \\ g(\tau_{opt}) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

We leverage Newton's numerical method again to solve this problem. We start the iteration of Newton's method with $A_0 = \begin{pmatrix} a^s_{opt} \\ \tau^s_{opt} \end{pmatrix}$. a^s_{opt} is the system-level factor and τ^s_{opt} represents the corresponding optimal checkpointing interval derived from sub-section III-B. The iterative function is $A_{n+1} = A_n - [J_F(A_n)]^{-1}F(A_n)$, where $J_F(A_n) = \begin{pmatrix} \frac{\partial f}{\partial a} & \frac{\partial f}{\partial \tau} \\ \frac{\partial g}{\partial a} & \frac{\partial g}{\partial \tau} \end{pmatrix}$ is the Jacobian Matrix. Similar as Algorithm 1, system-level factor a^s_{opt} should also be considered in deciding the optimal number of processes for the combined optimization.

D. Spare Node Allocation

Replacement-based failure recovery is a promising solution to mitigating the impact of failures on the performance since it is widely observed that spare nodes are available in production systems [17] [26]. Suppose we have a compute nodes in a system, in this sub-section we study the selection of b, the optimal number of spare nodes to tolerate failures from a compute nodes.

For the queueing system composed of $a + b$ nodes, we have the failure arrival rate of $\lambda = a \times \lambda_f$. Spare nodes do not have jobs and can be considered as failure-free. In this scenario the failure is recovered by physical repair because the failed nodes eventually need to be physically fixed to keep the sustainability of the system.

Let n be the random variable that reflects the number of concurrent failures to a nodes. Based on existing results from queueing theory [11], we have the mean and standard deviation (Std) of n as,

$$\begin{aligned} E(n) &= \lambda/\varphi + (\lambda/\varphi)^2(1+\sigma^2\varphi^2)/(2(1-\lambda/\varphi)) \\ &= \rho + \rho^2(1+\sigma^2\varphi^2)/(2(1-\rho)) \end{aligned} \tag{14}$$

$$\begin{aligned} Std(n) &= \sqrt{V(n)} \\ &= \sqrt{E(n) + \lambda^2\sigma^2 + \frac{\lambda^3 E(s^3)}{3(1-\rho)} + \frac{\lambda^4 E^2(s^2)}{4(1-\rho)^2}} \end{aligned} \tag{15}$$

Here $E(s^2)$ and $E(s^3)$ are the second and third moments of failure repair time respectively, which can be derived from φ, σ, the distribution and its moment generation function. Formulas (14) and (15) also reveal that ρ is an important factor to determine the number of concurrent failures.

We use $[E(n) - k \times Std(n), E(n) + k \times Std(n)]$ to curve the range of n and set $b_k = E(n) + k \times Std(n)$ spare nodes to tolerate the concurrent failures happen to the system. k is a small positive integer to control the accuracy and efficiency of spare node allocation. There is a tradeoff with the selection of k. A smaller k comes with conservative spare node allocation and a higher k corresponds to an aggressive approach of spare node allocation. The impacts with different values of k are discussed in sub-section V-C.

IV. Scalability Analysis

In Figure 2 we plot the optimal number of processes, checkpointing interval and the corresponding application execution time under different environments. In each sub-figure we first demonstrate the optimal performance for the default environment, where each parameter is set as the default values in Table I. Next we change the value of one parameter in each environment to observe its impact on the scalability. Coordination cost grows with the increase of the number of processes and has more sensitive impact on the scalability. We plot five series of bars with different coordination cost to study its impact. In addition to the first series of bars with default value of δ, in the second series we decrease the message passing overhead q to $1/10$ of the default value. We eliminate the coordination cost in the third series of bars to observe the coordination-free performance. The coordination cost can also be optimized by reducing the number of synchronization messages needed to achieve the global consistent state [13], which is elusive to model accurately since it largely depends on the number of processes that are involved in the communication. Based on the observation from [22], we assume the coordination cost for the improved protocol is proportional to $log(a)$ and reflect their performance in the last two series of bars.

In sub-figure 2a, the values for the number of optimal processes for the last three bars are the same for all the environments. This implies that it is system-level factor that determines the scalability if the coordination cost is sublinear to the number of processes. From the perspective of checkpointing, the optimization of coordination cost is the key to boost the scalability. When system-level factor limits the scalability, a checkpointing protocol with the complexity of $O(log(a))$ is good enough to match the third bar, which neglects the coordination cost.

The second series of bar($\delta = p + 0.00006 \times a$) with lower message passing cost also presents good scalability: its optimal number of processes is determined by the system-level factor for most environments. It will limit the optimal number of processes only if we increase the failure repair rate or the reliability of each node, as shown in the environments of $\varphi = 1$ and $MTBF = 32768$. In addition, we find that the second series of bar suffers from low efficiency. For example, when $\mu_f = 0.01$, all the last four series are limited by the system-level factor and share the same optimal number of processes. However, the application execution time for the second bar in sub-figure 2c is 234.85 hours, which is about 70 hours more than the other three series. Sub-figure 2b reveals the reason of the inefficiency: the checkpointing interval of the second bar is larger than others , which risks the application with more rework penalties.

I/O overhead p of checkpointing and failure recovery cost μ_f are less critical factors to determine the scalability, especially when they are already good enough. For example, there is little scalability improvement for all the series if we change the value of p from 0.05(default) to 0.005. Similar observations can be found if we reduce the failure recovery cost μ_f from 0.1(default) to 0.01 hours.

We highlight the following implications from the scalability analysis:

- It makes little sense to reduce failure recovery cost if it is already low enough, i.e. 0.1 hour, which is achievable with current fast recovery techniques [16].
- Coordination cost of checkpointing contributes significantly in limiting the scalability. Reducing the message passing overhead q is a compelling solution to break through the bottleneck. However, lowering q may make the nodes less effective for useful computation.
- With the current checkpointing implementations that are linearly affected by the number of processes, hardware factors, i.e., failure repair rate and MTBF on each node, are not critical in determining the scalability.

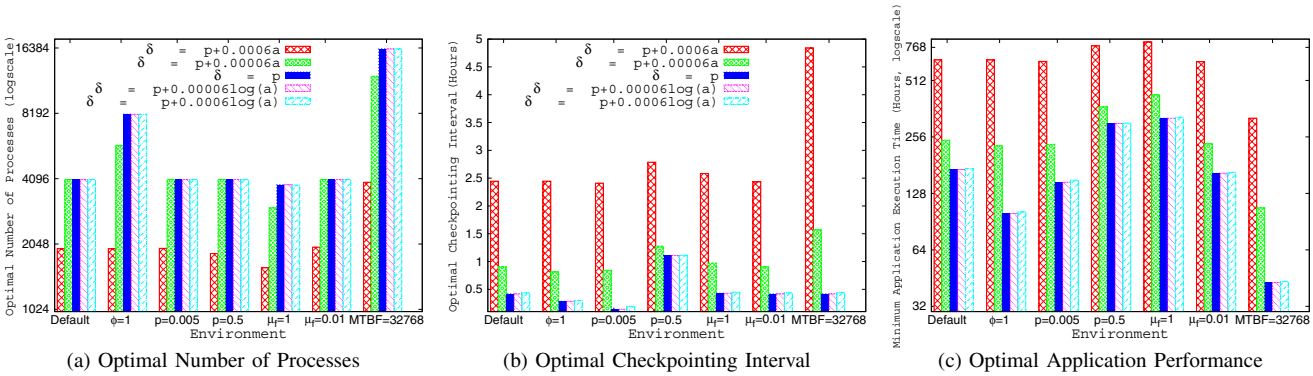
V. Experimental Study

A. Model Verification

Several environments are set up to verify the model. The *default* environment sets the parameters to the default values listed in Table I. We change the value of one parameter in other environments to observe the accuracy of the model under different scenarios. X-axis varies the value of a from 256 to 8196.

Failures arrivals are generated with the assumed exponential distribution and the parameters specified by each environment. Lognormal distribution is a good approximation for failure recovery and is used to generate the failure recovery log [19]. For each environment, we run the jobs for 10000 times at different job submission time and get the corresponding *simulated* mean and standard deviation of application execution time from the sample. We then use them to compare with the *predicted* mean and standard deviation computed from the model to verify the accuracy of the model.

Sub-figure 3a demonstrates the simulation results under six environments with the number of processes varied. We



(a) Optimal Number of Processes (b) Optimal Checkpointing Interval (c) Optimal Application Performance

Figure 2: Scalability Analysis

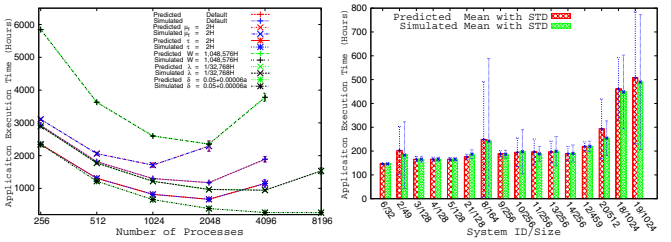
omit some points for the first three environments that suffer deadly from failures when the system size is sufficiently large. As expected, we observe that the predicted and simulated curves from the same environment are overlapped with each other. The differences between the simulated and predicted performance are trivial compared with the overall cost, if not negligible. The maximum gap we observed is 2 hours for limited scenarios, which is more than satisfactory.

An interesting question is whether the proposed model can be applied in real situations since it is observed in [19] that exponential distribution is not an ideal approximation of failure arrivals. To answer this question, we conduct experiments on the failure trace collected by the Los Alamos National Lab (LANL) [15].

We verify the model on all the systems that are sized 32 nodes or more. The sequential workload W is set to be $a \times 128$ hours. Physical failure repair time is logged by the trace, which is used as failure recovery cost. As shown in sub-figure 3b, each system is marked as one point on x-axis, with two error bars that reflect means and standard deviations for both predicted and simulated application performance.

Though not neatly precise as the data presented in sub-figure 3a, we still observe a close matching of the predicted and simulated application execution time in sub-figure 3b. As an example, for system 19 with 1024 nodes, the model estimates the application time accurately: the predicted and simulated means are 504.449 and 489.608 hours, respectively.

System 20 presents relatively more deviation than others. The model overestimates nearly 40 hours of the performance mean, which is not trivial compared with the actual mean execution time of 254 hours. Our analysis shows that the deviation is attributed to the large coefficient of variation of the failure recovery, which is $\theta_f = 21.85/3.58 \approx 6.1$ for system 20 and the largest among all the systems. Actually, the deviation is reduced to 12.7 hours by neglecting the failures with recovery cost more than 100 hours, which decreases the coefficient of variations to 2.44.



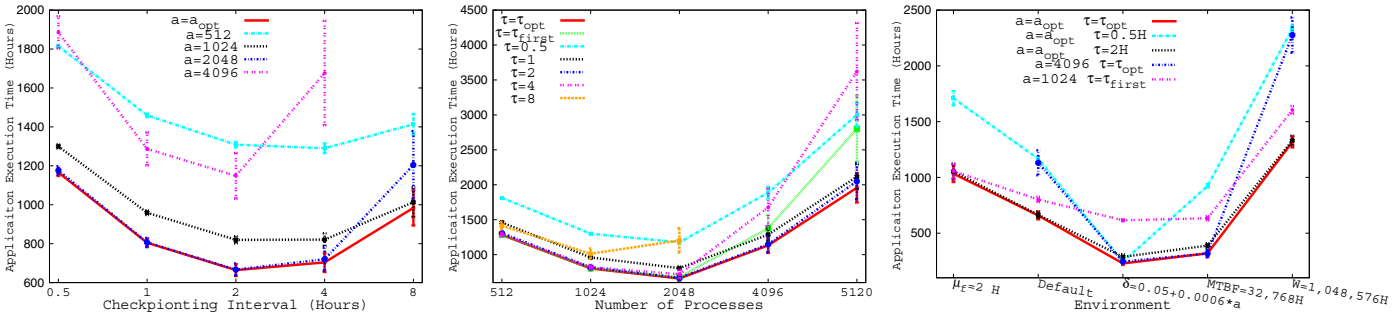
(a) Simulation Results (b) Experimental Results on LANL Data

Figure 3: Model Verification

B. Performance Optimization

We show the efficiency of the proposed performance optimization in Figure 4. In sub-figure 4a we curve the application execution time with different checkpointing intervals for a set of number of processes. Other parameters are set as defaults as listed in Table I. The first curve of $a = a_{opt}$ illustrates the application performance with optimal number of processes derived from our solution. We can observe that it outperforms other curves with significant advantages. The curve of 2048 processes is the closest to the optimal one when the checkpointing interval is 4 hours or less. However, its performance is degraded severely for the checkpointing interval of 8 hours. Though equipped with the largest number of processes, the curve of 4096 processes does not necessarily lead to the best performance.

Sub-figure 4b plots the application execution time with different number of processes for a set of checkpointing intervals. Again, the first curve where $\tau = \tau_{opt}$ reflects the optimal interval derived from our solution and leads to the best performance. It slightly outperforms the performance of the curve with checkpointing interval of 2 hours. The advantage is enlarged as the number of processes increases. The performance for the first order estimated optimal checkpointing interval is illustrated in the second curve. The performance of the first order estimation is decreased as the



(a) Optimal Number of Processes (b) Optimal Checkpointing Interval (c) Combined Optimization

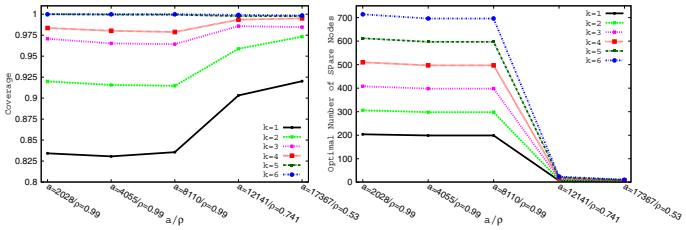
Figure 4: Optimization Verification

number of processes increases. This is due to the fact that more processes augments both $\lambda = a \times \lambda_f$ and $\delta = p + q \times a$, and invalidates the assumption of $\gamma\lambda \to 0$ made by the first order estimation.

The combined optimization is verified as shown in sub-figure 4c. We vary both a and τ, and set other parameters to the default values in the *default* environment. Other environments change one parameter and are marked at x-axis in the sub-figure. The first curve where $a = a_{opt}, \tau = \tau_{opt}$ reflects the combined optimization and always leads to the best performance compared with other single-side optimization approaches.

Another concern of Newton's method is the converge speed, which depends largely on the starting point. Our experiments reveal that l the optimizations terminate in less than 10 iterations when setting system-level factor and first order estimation as the starting points.

C. Spare Node Allocation



(a) Coverage with Different k (b) Optimal Number of Spare Nodes

Figure 5: Spare Node Allocation Verification

Figure 5 verifies the spare node allocation methodology. Each column on the x-axis represents a scenario with different number of processes and failure intensity. The process number of each scenario is actually the optimal number of processes for the checkpointing overhead of $\delta = 0.05 + 0.00006 \times a$, corresponding to the MTBF on each node varied from 4096, 8192, 16384, and 32768 to 65536, respectively. For the scenarios of $\rho = 0.99$, the optimal number of processes is determined by system-level factor. b_k for each scenario is calculated based on the approaches introduced in sub-section III-D.

We randomly select 10000 time spots and observe the number of failures happen to the system at each spot. We consider the spot as *covered* if its number of concurrent failures is less than or equal to b_k. We define

$$coverage = \frac{Number\ of\ covered\ spots}{10000} \tag{16}$$

to characterize the quality of b_k spare nodes.

In sub-figure 5a we plot the coverage for different k. We observe that the curves of $k = 3$ and $k = 4$ are featured with coverage more than 96% and 97%, respectively. However, we would like to suggest setting $k = 5$, which is highlighted with coverage more than 99.5% or even 100%. The advantage of $k = 6$ over $k = 5$ is trivial, if not negligible.

Sub-figure 5b illustrates the value of b_k for each scenario. One interesting observation is that intensity ρ has more impact on the optimal number of spare nodes than the number of compute nodes a. For example, although featured with different a, the number of spare nodes for the first three scenarios are almost the same. Moreover, the last two scenarios of each curve demonstrate a huge disparity in the values of a and b: they only need dozen spare nodes for more than 10000 compute nodes. The relatively low values of ρ for the last two scenarios explain this disparity. In consistence with formulas (14) and (15), this observation confirms that ρ is the key factor to determine the number of spare nodes. The spare node allocation depends little on the number of compute nodes involved in the application.

VI. Related Work

Understanding the impact of failures and fault-tolerant mechanisms has been an active research area for decades. Although the reliability of single node has been constantly improved, the large scale of current parallel systems puts forward many new challenges to the stage.

Young gave a solution to the optimal checkpointing interval selection by minimizing the total lost time due to failures and checkpointing [25]. Duda studied the impact of

checkpointing and failure repair on application performance in [5]. Garg et al. have proposed a checkpointing model to minimize the completion time of a program by assuming a general distribution of failure arrival in [8].

Daly made a big step towards a complete solution of optimal checkpointing interval in [4]. The author derived a more complete cost function and demonstrated a perturbation solution to an accurate high order approximation to the optimal checkpointing interval.

Our previous work of [24] [12] proposed a performance model under failures with uncoordinated checkpointing, where each process conducts checkpointing independently without reaching a global consistent state.

Plank et al. proposed to use spare processors for fault tolerance [18]. They modeled the application performance with a *Birth-Death* Markov Chain and derived an optimal number of spare nodes and checkpointing interval with a brute-force iterative solution. While their solution works well for small-scale systems, it is costly when the number of processes becomes large, such as in the scale of tens of thousands.

Elnozahy and Plank studied the impact of checkpointing for Peta-scale systems and discussed directions for achieving performance under failures in future high-end computing environments [7]. In [22], Wang et al. modeled coordinated checkpointing with *Stochastic Activity Networks* and studied the scalability, reliability and performance of the system. In [27], Zheng and Lan proposed reliability-aware scalability models to evaluate the performance of parallel systems and fault-tolerant techniques. [27] built the scalability model on the performance modeling of [4] [8].

Our work differs the existing research from the following aspects:

- *Performance Modeling*: We extend the assumption of failure recovery cost to general distribution, which improves existing work that considers failure recovery as fixed or exponentially distributed [25] [5] [8] [18] [4] [27]. According to [19], exponential distribution is not a good fit of failure recovery.
  Similar as [22] [4], we also generalize the first order assumptions. Our model allows failures to occur at any time even during the period of checkpointing and failure recovery.
  This study extends our previous work of [24] [12] to coordinated checkpointing environments, which dominate current parallel computing practice due to its advantage over *domino effects* and simplicity [6] [7] [20]. Distinguished from the work that assumed a fixed value [25] [5] [8] [4], we consider checkpointing overhead as a function of parameters such as I/O overhead, coordination cost and the number of processes.
- *Performance Optimization*: We improve [18] with a fast numerical solution to optimizing the number of processes and checkpointing interval. In addition, we introduce a solution to deriving the combined optimization of both number of processes and checkpointing interval, which is not considered in existing work. The queueing model of this research makes it feasible for an optimal spare node allocation, which is rarely studied in the existing work.
- *Scalability Analysis*: Our work differs from [7] [22] with an analytic approach to modeling the performance scalability, instead of simulation. This feature makes our model ready to use for other researches to study the performance, reliability and related topics with under failures and C/R. The target of our scalability analysis is performance optimization, instead of performance evaluation of [27]. Equipped with a more practical, general and accurate model, we believe this study has more potential in large-scale computing environment with hundreds of thousands of nodes.

VII. Conclusions and Future Work

New computing paradigms such as data center and Cloud computing push more and more HPC into the day-to-day life. Compared to traditional technical and engineering applications, these new computing paradigms require higher quality of service and productivity, in addition to computing power. In the meantime, with the increasing ensemble size, failure has become an inevitable factor of modern HPC systems. Performance under failure has become a critical issue facing the HPC community.

In this study, we have carried out a systematic study to model, evaluate, and optimize application performance under coordinated checkpointing fault-tolerant environments for modern large-scale parallel computing systems. We have first introduced a queueing based model to characterize and predict the application execution time under failures. Based on the newly proposed model, we next have presented performance optimization solutions to determining optimal checkpointing interval, optimal number of processes, and optimal number of spare nodes to minimize application execution time. The solution for a combined optimization of both the number of processes and checkpointing interval is also presented. Then, we have evaluated the importance of different parameters for a scalable computing environment under failures. Finally extensive experiments have been carried out based on both synthetic and actual failure logs. Experimental results show that both the proposed model and optimization algorithms work efficiently with satisfactory accuracy.

In the future we would like to incorporate the proposed model into existing fault-tolerant computing environments. We would also like to explore more performance optimization methodologies to mitigate the performance loss due to failures or failure handling. Overall, we plan to build a solid foundation for developing scalable fault-tolerant parallel computing environments.

ACKNOWLEDGMENT

This research was supported in part by National Science Foundation under NSF grant CCF-0937877, CNS-0834514, CNS-0751200, CCF-0702737, and by Department of Energy SciDAC-2 program under the contract No. DE-FC02-06ER41442.

REFERENCES

[1] F.S. Acton, Numerical Methods That Work, Chapter 2. *Mathematical Association of America*,1990.

[2] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier, etc, MPICH-V Project: A Multiprotocol Automatic Fault Tolerant MPI, *Proc. of International Journal of High Performance Comuting Applications*, 2006.

[3] K. M. Chandy and L. Lamport, Distributed Snapshots: Determining Global State of Distributed Systems. *ACM Transcations on Computer Systemss*, 3(1):63-75, 1985.

[4] J.T. Daly, A Higher Order Estimate of The Optimum Checkpoint Interval for Restart Dumps, *Future Generation Computer Systems*, Vol. 22 pp.301-312, 2006.

[5] A. Duda, The Effects of Checkpointing on Program Execution Time, *Information Processing Letters*, Vol 16, pp: 221-229, 1983.

[6] E. Elnozahy, L. Alvisi, Y.M. Wang and D.B. Jonson, A Survey of Rollback-Recovery Protocols in Message-Passing Systems, *ACM Computing Survey*, Sep 2002.

[7] E. Elnozahy and J. Plank, Checkpointing for Peta-Scale Systems: A Look into The Future of Practical Rollback-Recovery, *IEEE Trans. Dependable and Secure Computing*, 1-2: 97-108, 2004.

[8] S. Garg, Y. Huang, C. Kintala, and K. Trivedi, Minimizing Completion Time of a Program by Checkpointing and Rejuvenation, *Proc. of SIGMETRICS*,1996.

[9] J. Hursey, J. M. Squyres, T. I. Mattox and A. Lumsdaine, The Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI. *Workshop on Dependable Parallel, Distributed and Network-Centric Systems (DPDNS'07)*,2007.

[10] J. Hursey, T. Mattox and A. Lumsdaine, Interconnect Agnostic Checkpoint/Restart in Open MPI. *Proc. of 18th ACM International Symposium on High Performance Distributed Computing (HPDC'09)*, June, 2009.

[11] R. Jain, The Art of Computer Systems Performance Analysis, *Jone Wiley & Sons*, pp. 540-541. 1991.

[12] H. Jin, X.-H. Sun, Z. Zheng, Z. Lan and B. Xie, Performance under Failures of DAG-based Parallel Computing, *IEEE/ACM International Symposium on Cluster Coomputing and the Grid (CCGrid'09)*, June 2009.

[13] J.L. Kim, T. Park, An Efficient protocol for Checkpointing Recovery in Distributed Systems, *IEEE Trans. on Parallel and Distributed Systems*, 4(8): 955-960. 1993

[14] Z. Lan and Y. Li, Adpative Fault Management of Parallel Applications for High Performance Computing, *IEEE Trans. Computers*, 57(12): 1647-1660, 2008.

[15] Lo-s Alamos National Laboratory, Opertional Data to Support and Enable Computer Science Research, *http://institute.lanl.gov/data/lanldata.shtml*.

[16] Y. Li and Z. Lan, A Fast Recovery Mechanism for Checkpointing in Networked Environment, *Proc. of International Conference on Dependable Systems and Networks (DSN'08)*, June 2008.

[17] Y. Li, Z. Lan, P. Gujrati, and X.-H. Sun, Fault-Aware Runtime Strategies for High Performance Computing, *IEEE Trans. Parallel and Distributed Systems*, 20(4): 460-473, 2009.

[18] J. Plank and M. Thomason, Processor Allocation and Checkpointing Interval Selection in Cluster Computing Systems, *Journal of Parallel and Distributed Computing*, 61(11): 1570-1590, 2001.

[19] B. Schroeder and G. A. Gibson, A Large-Scale Study of Failures in High-Performance Computing Systems, *Proc. of International Conference on Dependable Systems and Networks (DSN'06)*, June 2006.

[20] S. Sankaran, J.M. Squyres, B. Barrett, etc. The LAM/MPI Checkpointing/Restart Frameworks: System-Initiated Checkpointing, *International Journal of High Performance Applications*, 19-4: 179-493, 2005.

[21] Top 500 Supercomputer Website. *http://www.top500.org*.

[22] L. Wang, K. Pattbbiraman, Z. Kalbarczyk, R. K. Iyer, L. Votta, C. Vick and A. Wood, Modeling Coordinated Checkpointing for Large-Scale Supercomputers, *Proc. of International Conference on Dependable Systems and Networks (DSN'05)*, June 2005.

[23] C. Wang, F. Mueller, C. Engelmann and S. L. Scott, A Job Pause Service under Lam/MPI+BLCR for Transparent Fault Tolerance, *Proc. of Parallel and Distributed Processing Symposium 2007 (IPDPS'07)*, March. 2007.

[24] M. Wu, X.-H. Sun, and H. Jin, Performance under Failures of High-End Computing, *Proc. of ACM/IEEE SuperComputing Conference 2007 (SC'07)*, Nov. 2007.

[25] J.W. Young, A first order approximation to the optimum checkpoint interval, *Communications of ACM*, Vol. 17 pp.530-531, 1974.

[26] Y. Zhang, M.S. Squillante, A.Sivasubramaniam and R. K. Sahoo, Performance Implications of Failures in Large-Scale Cluster Scheduling, *Proc. Workshops on Job Scheduling Strategies for Parallel Processing(JSSPP'04)*, pp.233-252, 2004.

[27] Z. Zheng and Z. Lan, Reliability-Aware Scalability Models for High Performance Computing, *Proc. of IEEE Cluster 2009*, Aug. 2009.

[28] Z. Zheng, Z. Lan, B-H. Park, and A. Geist, System Log Pre-processing to Improve Failure Prediction, *Proc. of International Conference on Dependable Systems and Networks (DSN'09)*, June 2009.