Optimizing HPC Fault-Tolerant Environment: An Analytical Approach

Hui Jin, Yong Chen, Xian-He Sun

Illinois Institute of Technology
{hjin6,chenyon1,sun}@iit.edu

Abstract

The increasingly large ensemble size of modern High-Performance Computing (HPC) systems has drastically increased the possibility of failures. Performance under failures becomes a timely important issue facing the HPC community. Checkpointing/Restart (C/R) is the widely used mechanism for fault tolerance, but is notorious for its cost, especially for large-scale parallel machines. We gauge three parameters, the number of compute nodes, the number of spare nodes, and checkpointing interval to conduct a comprehensive study of performance optimization under failures. Our 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. A numerical approach is used to derive both the optimal number of processes and the optimal checkpointing interval for best application performance. The optimal number of spare nodes for fault tolerance is also presented. In addition, performance scalability under failures is studied with upper bound analysis. Experimental results from both synthetic and actual system failure logs confirm that the proposed model and optimization methodologies are effective and feasible.

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 June 2009 are equipped with cores ranging from 51,200 to 194,912 [19]. 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 or even minutes [21] [13].

Checkpointing/Restart (C/R) is a widely used mechanism for fault-tolerant computing, where checkpointings are taken periodically to store a snapshot of the application state to a stable storage and used to restart the application in case of failures [4]. 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 increase failure arrival rate and checkpointing overhead.

In addition, two important decisions, the selection of the number of processes and the checkpointing frequency, are always faced by the users when they submit jobs for execution. The selection of larger number of processes can benefit the application by decreasing the workload on each node, but may introduces higher possibility of failures. Furthermore, it may also increase the checkpointing overhead for coordinated checkpointing, as the cost for achieving a consistent state is proportional with the number of processes. 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 when failures occur.

The cost of recovery is another important issue. Some researchers [15] [16] propose the usage of spare nodes to reduce failure recovery cost. 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 [12].

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 best performance. The contribution of this research is threefold:

- First, we propose a queueing model to predict application performance with consideration of workload, the number of processes, failure arrival rate, recovery cost, and checkpointing interval and overhead. We extend the modeling of failure recovery cost from fixed value or exponential distribution in existing work to general distribution. Our model also allows failures to occur during an on-going checkpointing or failure recovery, which matches actual engineering environments better.
- Second, we propose numerical approaches to deriving the optimal number of processes and optimal checkpointing interval for performance optimization based on the model. We also present a solution to determining the optimal number of spare nodes for performance optimization.
- Third, there is always a performance upper bound of a given application with the consideration of failures and checkpointing. For instance, increasing computational parallelism will decrease the workload per node, while increasing the failure arrival rate and checkpointing overhead. In other words, the performance may not be scalable even we assume the application is perfectly parallelizable. We study the upper bound of performance under 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 from different aspects, including the number of processes, checkpointing interval and the number of spare nodes. Upper bound 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 running on a homogenous large-scale cluster system. Sequential workload refers to the 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. The applications discussed in this paper are large-scale scientific applications that take days, weeks, or even months to finish the execution.

We assume the inter-arrival times of failures for each node are independent and identically distributed (*iid*) as exponential random variables with same parameter λ_f, which is the reverse of MTBF. The entire application will be halted if any process fails during the execution. It is a natural conclusion from probability theory that the failure arrivals of the parallel system with a nodes is also exponentially distributed with parameter $\lambda = a\lambda_f$.

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 [17], and we can reduce the recovery cost to several minutes, even seconds, by using the spare nodes [20] [21]. Regardless of the failure handling mechanism, we assume that the recovery time of each failure follows 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 has to be queued and will not be recovered until all the previous failures are fixed. A parallel application with a processes can be considered as an M/G/1 queuing system when we treat a failure as an event (customer) arrived in the queue.

Coordinated checkpointing is taken at fixed interval of τ. We model coordinated checkpointing overhead as $\delta = p+q\times a$, where p is the I/O overhead of image writing and $q\times a$ reflects the message passing overhead to get the consistent state [10]. A failure of any compute node will stall the entire parallel application. We have to rollback to the last checkpoint to resume the application after the failure is fixed. 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 in the *Appendix* 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 existing related work [10] [11] [17] [25].

B. Performance Model of a 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 is fixed, 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 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

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

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 content

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.

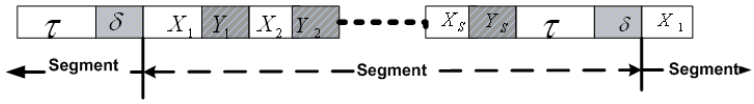


Fig. 1: Checkpointing Segment

X falls within the range of $0 \leqslant t < \gamma$ 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: $E(X) = \frac{1-e^{-\gamma\lambda}-\gamma\lambda e^{-\gamma\lambda}}{\lambda(1-e^{-\gamma\lambda})} = \frac{1}{\lambda} + \frac{\gamma}{1-e^{\gamma\lambda}}$, $V(X) = E(X^2) - E^2(X) = \frac{1}{\lambda^2} - \frac{e^{\gamma\lambda}\gamma^2}{(1-e^{\gamma\lambda})^2}$.

Under the assumption of the M/G/1 failure processing, we can get the first and second moments of Y using existing results from queuing theory [7], $E(Y) = \frac{\mu_f}{1-\lambda\mu_f}$, $E(Y^2) = \frac{\sigma_f^2+\mu_f^2}{(1-\lambda\mu_f)^3}$. The variance of Y is $V(Y) = E(Y^2) - E^2(Y) = \frac{\sigma_f^2+\lambda\mu_f^3}{(1-\lambda\mu_f)^3}$.

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 $E(S) = \frac{1-e^{-\gamma\lambda}}{e^{-\gamma\lambda}} = e^{\gamma\lambda} - 1$, $V(S) = \frac{1-e^{-\gamma\lambda}}{e^{-2\gamma\lambda}} = e^{\gamma\lambda}(e^{\gamma\lambda}-1)$.

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

$$\begin{aligned} E(T) &= E(E(T|S)) = E(\gamma + X_1 + X_2 + ... + X_S + Y_1 + Y_2 + ... + Y_S|S) \\ &= E(\gamma + SE(X) + SE(Y)) = (e^{\gamma\lambda}-1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda}) \end{aligned} \tag{2}$$

$$\begin{aligned} V(T) &= E(V(T|S)) + V(E(T|S)) = E(V(\gamma + U_X(S) + U_Y(S)|S)) + V(\gamma + SE(X) + SE(Y)) \\ &= E(S(V(X)+V(Y))) + V(S)(E(X)+E(Y))^2 \\ &= (e^{\gamma\lambda}-1)(\frac{1}{\lambda^2} - \frac{e^{\gamma\lambda}\gamma^2}{(1-e^{\gamma\lambda})^2} + \frac{\sigma_f^2+\lambda\mu_f^3}{(1-\lambda\mu_f)^3}) + e^{\gamma\lambda}(e^{\gamma\lambda}-1)(\frac{1}{\lambda} + \frac{\gamma}{1-e^{\gamma\lambda}} + \frac{\mu_f}{1-\lambda\mu_f})^2 \end{aligned} \tag{3}$$

Variance is a metric to describe the scale of a random variable. It complements mean and comprehensively characterizes a random variable [9]. Our model can derive both mean and variance of the application performance.

C. Performance of Parallel Applications

Each segment of $\gamma = \tau + \delta$ discussed in 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

$$\begin{aligned} E(\mathcal{T}) &= mE(T) + E(T') = \lfloor w/\tau \rfloor (e^{\gamma\lambda}-1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda}) + (e^{\alpha\lambda}-1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda}) \\ &\approx (w/\tau)(e^{\gamma\lambda}-1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda}) = (\frac{W}{a\tau})(e^{(\tau+p+qa)a\lambda_f}-1)(\frac{1}{a\lambda_f} + \frac{\mu_f}{1-\mu_f a\lambda_f}) \end{aligned} \tag{4}$$

The first term of $mE(T)$ reflects the mean time of successful m checkpointing segments and the second term of $E(T')$ is the mean time of the rest workload to finish the application.

Based on the assumption that the execution time of each individual segment is independent from that of another segment, we have the variance of $\mathcal{T}$ as,

$$
\begin{aligned}
V(\mathcal{T}) &= mV(T) + V(T') \qquad (5)\\
&= \lfloor w/\tau \rfloor ((e^{\gamma\lambda}-1)(\frac{1}{\lambda^2} - \frac{e^{\gamma\lambda}\gamma^2}{(1-e^{\gamma\lambda})^2} + \frac{\sigma_f^2 + \lambda\mu_f^3}{(1-\lambda\mu_f)^3}) + e^{\gamma\lambda}(e^{\gamma\lambda}-1)(\frac{1}{\lambda} + \frac{\gamma}{1-e^{\gamma\lambda}} + \frac{\mu_f}{1-\lambda\mu_f})^2)\\
&\quad + ((e^{\alpha\lambda}-1)(\frac{1}{\lambda^2} - \frac{e^{\alpha\lambda}\alpha^2}{(1-e^{\alpha\lambda})^2} + \frac{\sigma_f^2 + \lambda\mu_f^3}{(1-\lambda\mu_f)^3}) + e^{\alpha\lambda}(e^{\alpha\lambda}-1)(\frac{1}{\lambda} + \frac{\alpha}{1-e^{\alpha\lambda}} + \frac{\mu_f}{1-\lambda\mu_f})^2)
\end{aligned}
$$

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 novel model to predict the performance of parallel applications under failures, which considers a wide range of parameters including the number of processes, failure arrival rate, recovery time, checkpointing interval and overhead. The application execution time is monotonically increasing over parameters of failure arrival rate, recovery time and checkpointing overhead. We can always improve the application performance by lowering failure arrival rate, recovery time and checkpointing overhead with no exception. However, the selection of the number of processes, checkpointing interval and the number of spare nodes and their impact are more challenging and of great interest in terms of the performance optimization for parallel applications. The following sub-sections discuss the optimal selection of these 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 [7]. A system is referred as unstable if the number of failed nodes keeps increasing. Otherwise the system is stable.

Two factors decide the selection of the number of processes for a parallel application: *System-Level Factor* a^s_{opt} and *Application-Level Factor* a^a_{opt}. 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 best application performance. We have $a_{opt} = min\{a^s_{opt}, a^a_{opt}\}$ to guarantee both the system stability and the best application performance.

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 reverse of mean failure repair time. Please note that the value of φ is decided 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 denote $\rho = \lambda/\varphi = a\lambda_f/\varphi$ as the *failure intensity* of the system.

Based on queuing 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 $a^s_{opt} = \frac{0.99\varphi}{\lambda_f}$.

a^s_{opt} imposes the maximum number of nodes that can be used for computation from the system's perspective. The summation of the number of compute nodes occupied by all applications running on the system cannot exceed a^s_{opt}. Otherwise the system will sooner or later exhaust the available healthy nodes and become unstable.

Algorithm 1 Find the Upper Bound of the Number of Processes

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

$a^s_{opt} \leftarrow \frac{0.99\varphi}{\lambda_f}$
$a^a_{opt} \leftarrow a^s_{opt}$
while $|g(a^a_{opt})| > \varepsilon$ **do**
  $a^a_{opt} \leftarrow a^a_{opt} - \frac{g(a^a_{opt})}{g'(a^a_{opt})}$
end while
$a_{opt} \leftarrow min\{a^s_{opt}, a^a_{opt}\}$

2) Application-Level Factor: Considering the expected application execution time of formula (4) 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 upper bound of a numerically.

Algorithm 1 demonstrates the pseudocode to find the optimal number of processes. 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 for the application.

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, which confirms the existing research results [2] [23]. Next we present a numerical methodology to estimate the optimal checkpointing interval.

1) First Order Approximation of Optimal Checkpointing Interval: 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 with $\gamma\lambda \to 0$:

- $\lim_{\gamma\lambda\to 0} E(X) = \lim_{\gamma\lambda\to 0}(\frac{1}{\lambda} + \frac{\gamma}{1-e^{\gamma\lambda}}) = \frac{\gamma}{2}$, which is confirmed by the research result from [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,

$$E(\mathcal{T}) = \frac{w}{\tau}E(T) = \frac{w}{\tau}(\gamma + SE(X) + SE(Y)) = \frac{w}{\tau}(\gamma + (e^{\gamma\lambda} - 1)(\frac{\gamma}{2} + \frac{\mu_f}{1-\lambda\mu_f}))$$
$$= \frac{w}{\tau}(\gamma + \gamma\lambda(\frac{\gamma}{2} + \frac{\mu_f}{1-\lambda\mu_f})) = \frac{w}{\tau}(\tau + \delta + (\tau+\delta)\lambda(\frac{\tau+\delta}{2} + \frac{\mu_f}{1-\lambda\mu_f}))$$

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 have the first order approximation of checkpointing interval as

$$\tau_{first} = \sqrt{2\delta(\frac{1}{\lambda} + \delta/2 + \frac{\mu_f}{1-\lambda\mu_f})} \approx \sqrt{2\delta(\frac{1}{\lambda} + \frac{\mu_f}{1-\lambda\mu_f})} \tag{6}$$

This result is consistent with the conclusion from [2]. It is also identical to the conclusion of [23] if the failure recovery cost is considered to be zero, as assumed by [23].

Algorithm 2 Find the Optimal Checkpointing Interval

Definition: $f(\tau) \equiv 1 - e^{\delta\lambda}e^{\tau\lambda}(1-\tau\lambda)$. ε is a postive real number sufficiently close to zero.
Objective: Find the optimal checkpointing interval.

$\tau_0 \leftarrow \sqrt{2\delta(\frac{1}{\lambda} + \frac{\mu_f}{1-\lambda\mu_f})}$
while $|f(\tau_0)| > \varepsilon$ **do**
  $\tau_0 \leftarrow \tau_0 - \frac{g(\tau_0)}{g'(\tau_0)}$
end while
$\tau_{opt} \leftarrow \tau_0$

2) Numerical Approach to Deriving Optimal Checkpointing Interval: In III-B.1 we approximate the optimal checkpointing interval with the assumption that $\gamma\lambda$ is negligible as zero. This assumption will not hold for 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. From formula (4), we have

$$\frac{\partial E(\mathcal{T})}{\partial \tau} = \frac{\partial (w/\tau)(e^{\gamma\lambda}-1)(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda})}{\partial \tau} = w(\frac{1}{\lambda} + \frac{\mu_f}{1-\mu_f\lambda})(\frac{1}{\tau^2})(1 - e^{\delta\lambda}e^{\tau\lambda}(1-\tau\lambda)) = 0.$$

By solving equation $1 - e^{\delta\lambda}e^{\tau\lambda}(1-\tau\lambda) = 0$, we have the optimal checkpointing interval. There is no exact mathematical solution to this equation and Newton's Method is adopted to solve it. The pseudocode to achieve optimal checkpointing interval with Newton's method is similar to Algorithm 1 and is illustrated in Algorithm 2. First order estimation is used as the starting point of the iteration.

C. Combined Optimization

In sub-section III-A we have presented a methodology to achieve the optimal number of processes with the assumption that all other parameters are known. Similarly, III-B demonstrates the derivation of optimal checkpointing interval with the knowledge of all other parameters. 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, as the number of processes can be ranged from dozens to hundreds of thousands. Next we show a numerical solution to approximating the optimal selection of the combination of (a, τ) efficiently.

Let $A_n = \begin{pmatrix} a_n \\ \tau_n \end{pmatrix}$, $f(a) \equiv \frac{\partial E(\mathcal{T})}{\partial a}$, $g(\tau) \equiv \frac{\partial E(\mathcal{T})}{\partial \tau}$ 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 use Newton's method to solve this system. We start 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 Algorithm 2. 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. We iterate till the function converges and get an optimal solution to the problem.

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 [12] [24]. 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 [7], we have the mean and standard deviation (Std) of n as,

$$E(n) = \lambda\mu_f + (\lambda\mu_f)^2(1 + (\frac{\sigma_f}{\mu_f})^2)/(2(1-\lambda\mu_f)) = \rho + \rho^2(1 + (\frac{\sigma_f}{\mu_f})^2)/(2(1-\rho)) \tag{7}$$

$$Std(n) = \sqrt{V(n)} = \sqrt{E(n) + \lambda^2\sigma_f^2 + \frac{\lambda^3 E(s^3)}{3(1-\rho)} + \frac{\lambda^4 E^2(s^2)}{4(1-\rho)^2}} \tag{8}$$

Here $E(s^2)$ and $E(s^3)$ are the second and third moments of failure repair time respectively, which can be derived from μ_f, σ_f, the distribution and its moment generation function. Formulas (7) and (8) also reveal that ρ is an important factor to decide the number of concurrent failures.

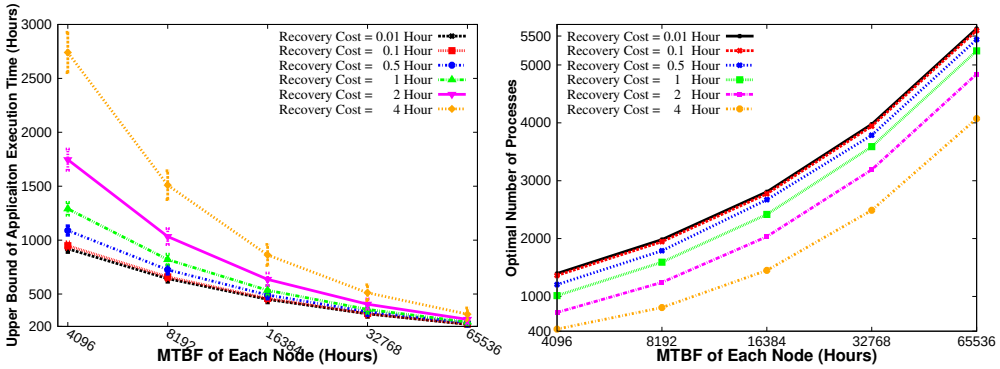
We use $[E(n) - k \times Std(n), E(n) + k \times Std(n)]$ to curve the range of n and set $b_k = E(n) + k \times Std(n)$ spare nodes to tolerate the concurrent failures happen to the system. k is a small positive integer to control the accuracy and efficiency of spare node allocation. There is a tradeoff 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. We will discuss the effects of different k in the experimental study of Section V.

IV. Upper Bound Analysis and The Implications

The optimal number of processes derived in Section III reflects the best performance we can get for an application and characterizes the scalability of an application-machine system. It does not help improve the performance of the application by assigning more compute nodes than the optimal number a_{opt}. In this Section we quantitatively analyze the performance upper bound of the application and the corresponding number of processes under different system parameters, and discuss the potential solutions to breaking the scalability bottleneck .

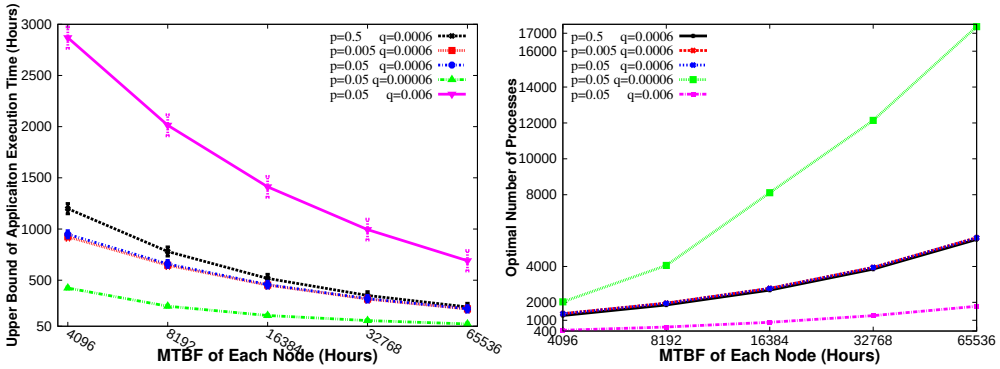
A. Evaluation Methods

We first obtain an optimal combination of a_{opt} and τ_{opt} based on sub-section III-C for an application with known λ_f, μ_f, and checkpointing overhead parameters p and q. Once a_{opt} and τ_{opt} are derived, the corresponding $E(\mathcal{T})$ and $Std(\mathcal{T}) = \sqrt{V(\mathcal{T})}$ can be achieved by using formula (4) and (5), which are actually the mean and standard deviation of the performance upper bound. We derive the optimal number of processes and application execution time with varied MTBF, μ_f and δ, plot them in Figure 2 and Figure 3, and analyze the implications. Other parameters are set to the default values in Table I if not specified explicitly.



(a) Upper Bound of Application Execution Time (b) Optimal Number of Processes

Fig. 2: Upper Bound with Different MTBF and Recovery Cost



(a) Upper Bound of Application Execution Time (b) Optimal Number of Processes

Fig. 3: Upper Bound with Different MTBF and Checkpointing Overhead

B. Upper Bound Analysis

1) Upper Bound for Different MTBF and Recovery Cost: Figure 2a illustrates the upper bounds of application performance with different MTBF and recovery cost. The corresponding optimal numbers of processes are plotted in Figure 2b.

Our first observation is that there is an upper bound limit for the number of processes we can utilize for a parallel application. If the MTBF of each node is 65,536 hours (about 7.5 years) and the recovery overhead is 0.01 hour (36 seconds), we have the upper bound of 5,628 nodes to achieve the maximum performance, which means increasing the number of nodes beyond this scale does not help improve the application performance any more. Please note that a node with MTBF of 7.5 years can be considered as ultra reliable and is rare in practice. We demonstrate this data here to find the maximum upper bound we can get without the restriction from failure arrival rate. The second observation is that, while impressed by the performance improvement by decreasing the failure recovery cost from 4 hours to 2 or 1, we find that the curves of the recovery cost with 0.01 hour and 0.1 hour are almost overlapped with each other for both Figure 2a and 2b. This observation reveals that it does not make a significant sense to improve the recovery cost if it is already fast enough.

Though the upper bound can be improved by increasing the MTBF or decreasing the failure recovery cost, our study indicates that neither MTBF nor recovery cost dominates the scalability bottleneck: the application cannot be benefited from tens of thousands of nodes no matter what values the MTBF and recovery cost are.

2) Upper Bound for Different MTBF and Checkpointing Overhead: The upper bounds with different MTBF and checkpointing overhead of application performance and the optimal number of processes are shown in Figure 3. Each curve has its own p and q values to represent one pattern

of checkpointing overhead.

The first three curves almost overlap with each other in Figure 3b. They all share one unique value of q as 0.0006. This fact implies that p contributes little to decide the optimal number of processes. In Figure 3a, the curve of $p = 0.5, q = 0.0006$ deviates a little bit from the other two overlapped curves. The deviation shrinks with the increase of MTBF.

An exciting observation of Figure 3 is that the fourth curve of $p = 0.05, q = 0.00006$ presents impressive scalability performance than others due to its significant low value of q: it can even use more than 10,000 nodes when the MTBF of each node is 32,768 hours or higher. When MTBF of each node is 16,384 hours or less for curve $p = 0.05, q = 0.00006$, we have $a^s_{opt} < a^a_{opt}$ and the optimal number of processes is decided by the system-level factor. The value of a^a_{opt} increases faster as the MTBF goes up and finally exceeds a^s_{opt} to become the dominant factor that decides the optimal number of processes.

C. Implications

The upper bound analysis has shown that q is the most important factor that decides the scalability under failures. It is crucial to develop more efficient and scalable checkpointing protocol and system with low-cost message communication to make the applications benefit from current large-scale systems with hundreds of thousands of nodes. Recovery cost of 0.1 hour is sufficient to get the best performance. Furthermore, there is no motivation to move beyond this point: the scalability improves little with a recovery cost less than 0.1 hour.

V. Experimental Study

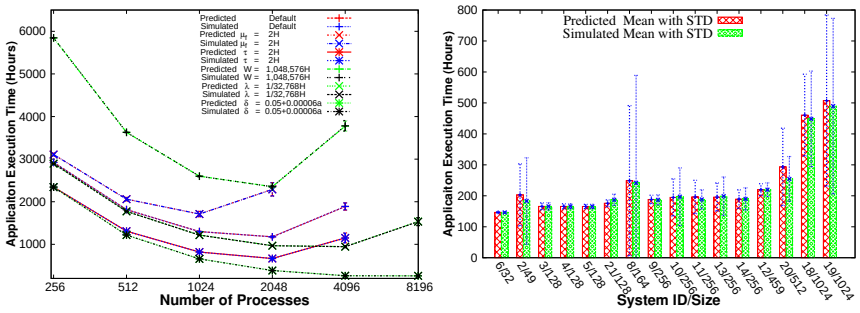
A. Model Verification

1) Simulation Results: 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. The value of a, the number of processes for an application is not fixed and is used to verify the model under different system sizes.

Each environment is basically a series of time events, such as job arrival, failure arrival, failure fixed, checkpointing started or finished, etc. 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 [17]. For each environment, we run the jobs for 10,000 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.

Figure 4a demonstrates the simulation results under six environments with the number of processes varied. We omit some points for the first three environments which suffers deadly from failures when the system size is sufficiently large. For example, the system intensity ρ is equal to or larger than 1 for the second environment of failure recovery mean $\mu_f = 2$ hours when the system size scales up to $a = 4,096$. In this case the system is unstable and it is meaningless to do performance prediction.

As expected, we observe that the predicted and simulated curves from the same environment are overlapped with each other, which confirms the correctness of the proposed model. It is also observed that a lower checkpointing overhead δ always leads to the best performance. Its advantage accumulates as the the number of processes increases, which supports the conclusion of Section IV well.



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

Fig. 4: Model Verification

2) Experimental Results: An interesting question is whether the proposed model can be applied in real situations since it is observed in [17] 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) [10]. The LANL failure trace is collected from 22 high-performance computing systems between year 1996 and November 2005. Most of these systems are large clusters of NUMA or SMP nodes.

We verify the model on all the systems that are sized 32 nodes or more. To guarantee the workload on each node to be 128 hours, the sequential workload W is set to be $a \times 128$ hours. As shown in Figure 4b, 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.

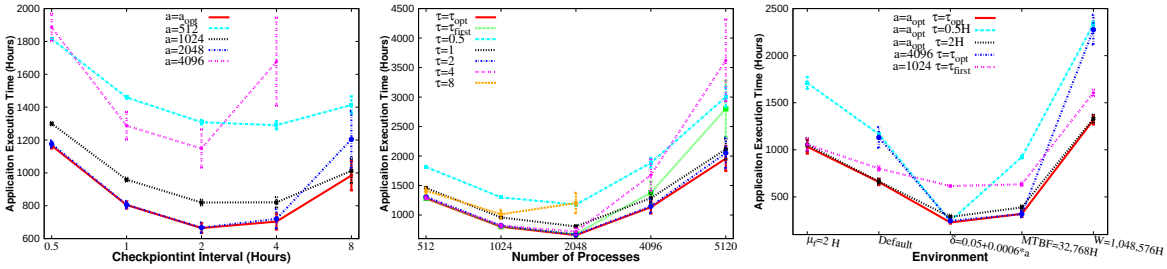
Though not neatly precise as the data presented in Figure 4a, we still observe a close matching of the predicted and simulated application execution time in Figure 4b. As an example, for system 19 with 1024 nodes, the model estimates the application time accurately: the predicted and simulated means are 504.449 and 489.608 hours, respectively. The standard deviations are also predicted with a high accuracy.

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 accordingly to 2.44.

The coefficient of variation of failure recovery θ_f may not only hurt the accuracy, but also lead to a high variation of application performance if the failure recovery cost is also high. For example, system 8 is featured with $\mu_f = 35.5683$, $\sigma_f = 100.475$, which leads to a coefficient of variations as $\theta_f = 2.82$, and a significantly large standard deviation of application execution time.

B. Performance Optimization

We show the efficiency of the proposed performance optimization in Figure 5. In Figure 5a 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 2,048 processes is the closest to the optimal one when the checkpointing interval is 4 hours or less. However, its performance is degraded seriously for the checkpointing interval of 8 hours. Though equipped with the largest number of processes, the curve of 4096 processes does



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

Fig. 5: Optimization Verification

not necessarily lead to the best performance. It is actually one of the worst, which confirms our discussion in Section IV.

Figure 5b 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 accuracy of the first order estimation is decreased as the 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 $\gamma\lambda \to 0$, on which the first order approximation is based.

The combined optimization is verified as shown in Figure 5c. 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 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. During the experiments, we optimize the number of processes from system-level upperbound a^s_{opt}. The checkpointing interval is optimized from the first order estimation. For the combined optimization, we first set $a = a^s_{opt}$, get the corresponding optimal checkpointing interval and start from there. All the optimizations terminate in less than 10 iterations in the experiment, which is considered quite satisfactory in terms of the converge speed.

C. Spare Node Allocation

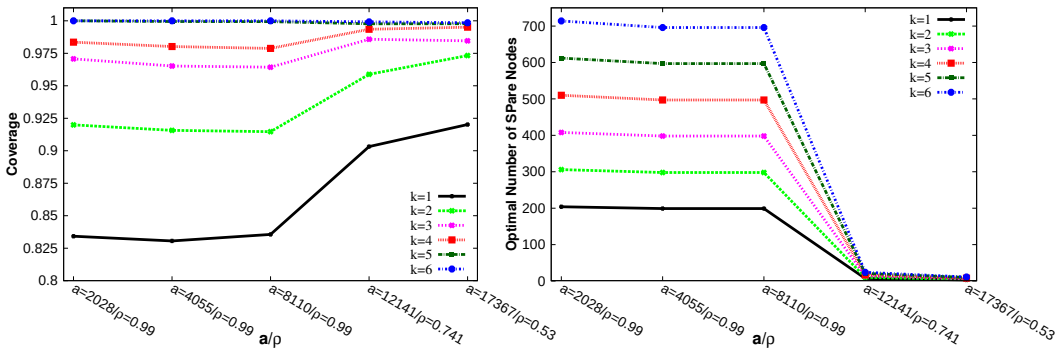
In Section IV we have observed that q is the dominant factor of scalability. The curve of $\delta = 0.05 + 0.00006 * a$ in Figure 3b is the only one to break though the upper bound of 10,000 processes and has a high demanding for an optimal spare node allocation to guarantee its performance. We use this curve as *base curve* to verify the spare node allocation.

Each point of the base curve corresponds to a specific optimal number of processes a, which is used to calculate b_k for $1 \leq k \leq 6$ based on sub-section III-D. We randomly select 10,000 times 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 have

$$coverage = \frac{Number\ of\ covered\ spots}{10,000}$$

to characterize the quality of b_k spare nodes.

In Figure 6a we plot the coverage for different k. The x-axis is marked with the value of a and system intensity ρ corresponding to each point of the base curve. We observe that the curves of $k = 3$ and $k = 4$ are featured with coverage more than 96% and 97%, respectively. However,



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

Fig. 6: Spare Node Allocation Verification

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.

Figure 6b illustrates the values of b_k corresponding to each point of the base curve. One interesting observation from this figure is that intensity ρ has more impact on the optimal number of spare nodes than the number of compute nodes a. For example, though featured with different a, the number of spare nodes for the first three points of each curve are almost the same. Moreover, the last two points of each curve demonstrate a huge disparity in the values of a and b: they merely need dozen spare nodes for more than 10,000 compute nodes. The relatively low values of ρ for the last two points explain this disparity. In consistence with formulas (7) and (8), this observation confirms that ρ is the key factor to decide 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. Though 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 [23]. However, this solution does not consider the failure recovery cost. Duda studied the impact of checkpointing and failure repair on application performance [3]. A limitation of his model is the assumption that the failure repair time follows exponential distribution. Garg and Huang have proposed a checkpointing model to minimize the completion time of a program by assuming a general distribution of failure arrival [6], which does not consider concurrent failures in the system.

Daly made a big step towards a complete solution of optimal checkpointing interval in [2]. The author derived a more complete cost function and demonstrated a perturbation solution to an accurate high order approximation to the optimal checkpointing interval. One limitation of [2] is that it simply assumes the values of failure recovery cost and checkpointing overhead are fixed, which does not fit parallel computing scenario well.

Our previous work of [22] [8] 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 [16]. 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 [5]. However, the conclusions were made based on simulations and cannot be used to model and predict the performance. In [25], Zheng and Lan proposed reliability-aware scalability models to evaluate the performance of parallel systems and fault-tolerant techniques. [25] built the scalability model on the performance modeling of [2] [6], and cannot step over their limitations as a result. This fact may limit its potential in large-scale environment.

Our work differs the existing research from the following aspects:

- *Failure Recovery Cost*: We extend the assumption of failure recovery cost to general distribution, which improves existing work that considers failure recovery as fixed or exponentially distributed [23] [3] [6] [2] [16] [25]. According to [17], exponential distribution is not a good fit of failure recovery.
- *Failure Occurrence*: Similar as [2], 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. Moreover, we consider the failures are served on a FCFS basis if they are overlapped, which is a reasonable assumption for current parallel systems, especially when the physical repair from a system administrator is involved in the failure recovery.
- *Checkpointing*: We extend the previous work of [22] [8] to coordinated checkpointing environments, which dominate current parallel computing practice due to its advantage over *domino effects* and simplicity [4] [5] [18]. In addition, rather than a fixed value, we consider checkpointing overhead as a function of parameters such as I/O overhead, checkpointing protocol and the number of processes,etc.
- *Performance Optimization*: We improve [16] with a fast numerical solution to optimizing the number of processes and checkpointing interval. In addition, we give 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.
- *Upper Bound Analysis*: Our work differs [5] with an analytic approach to modeling the performance scalability, instead of simulation. The target of our upper bound analysis is performance optimization, instead of performance evaluation of [25]. 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. Conclusion 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. This model distinguishes the impact of workload, the number of processes, failure arrival rate, recovery cost, and checkpointing interval and overhead. The mean application execution time and its variance are derived from the model. 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 spares 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 conducted an upper bound analysis to study the best performance possible under the occurrence of failures. We have illustrated the importance of scalable and effective checkpointing mechanisms for scalable fault tolerant computing. 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 also would 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.

REFERENCES

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

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

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

[4] E.N. 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.

[5] 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.

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

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

[8] 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.

[9] A. Kagan, L. A. Shepp, Why The Variance, *Statistics & Probability Letters*, 38(4), pp. 329-333, July 1998.

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

[11] Los Alomos National Laboratory, Opertional Data to support and Enable Computer science Researc, *http://institute.lanl.gov/data/lanldata.shtml*.

[12] 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.

[13] C.-D. Lu, Scalable Diskless Checkpointing. for Large Parallel Systems, *Ph.D dissertation, Department of Computer Science. University of Illinois at Urbana-Champaign*, 2005.

[14] V.F. Nicola, V.G. Kulkarni, and K.S. Trivedi, Queueing Analysis of Fault-Tolerant Computing Systems, *IEEE Trans. Software Engineering*, 13(3): 363-375, 1987.

[15] J. Plank, K. Li and M.A. Puening, Diskless Checkpointing, *IEEE Trans. Parallel and Distributed Systems*, 9(10):972-986, 1998.

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

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

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

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

[20] 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.

[21] C. Wang, F. Mueller, C. Engelmann and S. L. Scott, Proactive Process-Level Live Migration in HPC Environment, *Proc. of ACM/IEEE SuperComputing Conference 2008 (SC'08)*, Nov. 2008.

[22] 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.

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

[24] 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.

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

APPENDIX

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	$\frac{1}{8{,}192 Hours}$	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	$\delta = p + q * a$	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	$\gamma = \tau + \sigma$	Failure-free period needed by a successful checkpointing segment.
T	N/A	Random variable to finish a segment.
X_i or X	N/A	Random variable that reflects rework cost in terms of hours.
Y_i or 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, reverse of failure repair time.
ρ	λ/φ	System-level failure intensity. Must be less than 1 to guarantee the stability of the system.
a^s_{opt}	$\frac{0.99\varphi}{\lambda_f}$	System-level factor to decide the optimal number of compute nodes.
a^a_{opt}	N/A	Application-level factor to decide the optimal number of processes.
a_{opt}	$min\{a^s_{opt}, a^a_{opt}\}$	Optimal the number of processes corresponding to the best application performance.
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.