Shadow Computing

An Energy-Aware Resiliency Scheme for High Performance Computing

Bryan Mills[*], Taieb Znati[*†], Rami Melhem[*]
Department of Computer Science[*]
Telecommunications Program[†]
University of Pittsburgh
Pittsburgh, PA 15260
(bmills, znati, melhem)@cs.pitt.edu

ABSTRACT

Current fault-tolerance frameworks are designed to deal with both physical and logical fail stop errors and typically rely on the classic checkpoint-restart approach for recovery. The proposed solutions differ in their design, the type of faults they manage and the fault tolerance protocol they use. The major shortcoming of checkpoint-restart stems from the potentially prohibitive costs, in terms of execution time and energy consumption. Since faults are both voluminous and diverse, the inherent instability of future large-scale high performance computing infrastructure calls for a wholesale reconsideration of the fault tolerance problem and the exploration of radically different approaches that go beyond adapting or optimizing existing checkpointing and rollback techniques. To this end, we propose Shadow Computing, a novel energy-aware computational model to support scalable fault-tolerance in future large-scale high-performance computing infrastructure. We present three techniques for applying shadow computing to high performance computing: energy optimal replication, stretched replication, and minimum work replication. All of the proposed schemes provide the same fault-tolerance as pure replication while providing significant energy savings.

1. INTRODUCTION

To enable future scientific breakthroughs and discoveries, the next-generation of scientific applications will require exascale computing performance to support the execution of predictive models and the analysis of massive quantities of data, with orders of magnitude higher resolution and fidelity than what is possible in existing computing infrastructure [22, 16]. In order to deliver exascale computing and effectively harness its capabilities, several daunting scalability challenges must be addressed. In the late 90's, terascale performance was achieved with fewer than 10,000 single-core processors. A decade later, petascale performance required about 10 times as many processors as terascale performance. If such trends continue, delivering exascale computing will require a million processors, each supporting 1000 cores, resulting in a billion-core computing infrastructure with a massive increase in the number of memory modules, communications devices and storage components. Beyond raw computing, communications and storage requirements, future exascale computing systems are faced with unprecedented energy and resiliency challenges.

Power consumption is widely recognized as one of the most significant challenges facing exascale computing [16, 22]. The expected energy consumption increase in exascale computing is staggering. If current trends hold true, an exascale computing system is expected to consume over a gigawatt of power. This represents a 10 fold increase in the energy consumption of today's largest data centers, which typically ranges between 100 and 200 Megawatts. Reducing the power consumption of an exascale computing system to the DoE's target of 20 Megawatts is undoubtedly a formidable challenge, with a pervasive effect on next generation scientific applications. Addressing such a challenge requires building power and energy awareness into the foundations of future exascale computing infrastructure, with "performance per watt" as the metric of merit to measure efficiency. Radical approaches must be developed to achieve efficient energy management across all hardware and software components of the system.

In addition to its impact on energy consumption, the upward trend in the number of computing nodes also has a direct negative effect on the overall system reliability. Even if the individual node failure rate is low, the overall system failure rate quickly becomes unacceptable as the number of components increases. For example, a computing system with 200,000 nodes will experience a mean time between failure(MTBF) of less than one hour, even when the MTBF of an individual node is as large as 5 years [21]. The dramatic decrease in system reliability as the number of computing nodes increases is depicted in Figure 1 [21]. Other factors are also expected to increase failure rates in exascale computing, including the expected high fault rates of advanced-technology computing components operating at lower voltage levels and the impact of undesirable aging effects as they become significant[23]. Addressing these concerns brings about unprecedented resiliency challenges, which puts in question the ability of next generation high performance computing to continue operation in the presence of faults without compromising the requirements of the supported applications.

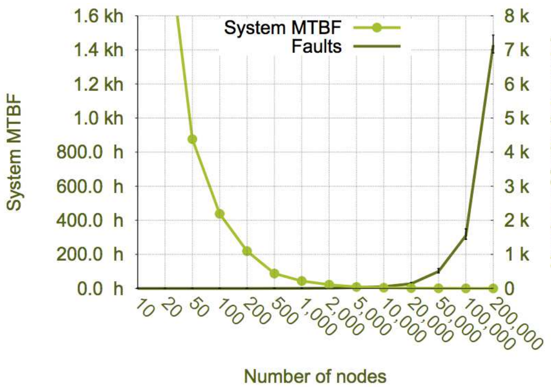


Figure 1: Effect on system MTBF as number of nodes increase.

The current response to faults consists of restarting the execution of the application, including those components of its software environment that have been affected by the occurring fault. To avoid the full re-execution of the failing application, fault-tolerant techniques typically checkpoint the execution periodically; upon the occurrence of a hardware or software failure, recovery is achieved by restarting the computation from a safe checkpoint. In some situations, however, several components of the software environment associated with the failed application may have to be restarted.

Given the anticipated increase in failure rate and the time required to checkpoint large-scale compute- and data-intensive applications, it is very likely that the time required to periodically checkpoint an application and restart it upon failure may exceed the mean time between failures. Consequently, applications may achieve very little computing progress, thereby reducing considerably the overall performance of the system. For example, a study carried out at Sandia National Laboratories focused on evaluating the overhead incurred by checkpointing in exascale computing environments. The results of the study, depicted in Figure 2, clearly show that beyond 50,000 nodes the application spends only a fraction of the elapsed time performing useful computation.

The main objective of this paper is to explore radically different paradigms to enable scalable resiliency with minimum energy consumption in the future exascale computing infrastructure. To this end, we propose a new energy-aware "shadow computing" model, as the basis for an efficient and scalable computational framework to achieve desired levels of fault tolerance, while minimizing energy consumption. The proposed model goes beyond traditional check-pointing and roll-back recovery techniques, and uses a multi-level, energy-aware replication approach to achieve scalable fault tolerance in exascale computing.

The basic idea of the shadow computing model is to associate with each process a suite of "shadow processes", whose size depends on the "criticality" and performance requirements of the underlying application. A shadow is an exact replica of the main process. In order to overcome failure, the shadow is scheduled to execute concurrently with the main process,

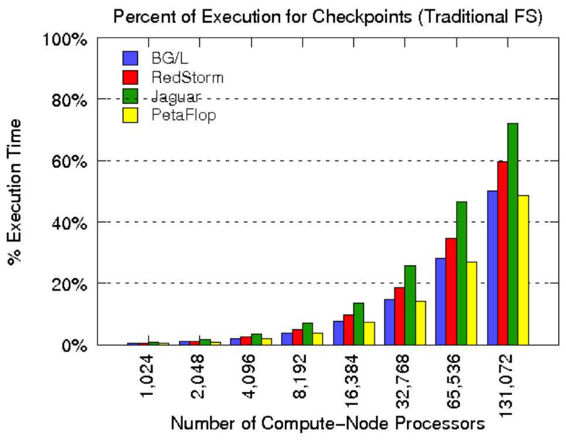


Figure 2: Percent of time to perform checkpoints.

but at a different computing node. Furthermore, in order to minimize energy, shadow processes initially execute at decreasingly lower processor speeds. The successful completion of the main process results in the immediate termination of all shadow processes. If the main process fails, however, the primary shadow process immediately takes over the role of the main process and resumes computation, possibly at an increased speed, in order to complete the task. Moreover, one among the remaining shadow processes becomes the primary shadow process.

It is worth noting that, since the failure of an individual component is much lower than the aggregate system failure, it is very likely that most of the time the main processes complete their execution successfully. Successful completion of a main process automatically results in the immediate halting of its associated shadow processes, with a significant saving in energy consumption. Furthermore, the number of shadow processes to be instantiated in order to achieve the desired level of fault-tolerance must be determined based on the likelihood that more one process failure within the execution time interval of the main task.

The main challenge in realizing the potential of the shadow computing model stems from the need to compute the speed of execution of the main process and the speed of execution of its associated shadows, both before and after a failure occurs, so that the target response time is met, while minimizing energy consumption. Given the nature of failure in exascale computing, it is unlikely that both the main and its primary shadow fail simultaneously. Therefore, only a dual level of redundancy, whereby only one shadow process is executed concurrently with the main process, is needed in most cases to achieve the desired level of fault-tolerance. The completion of a main or its shadow results in the successful execution of the underlying task.

The main contributions of this paper are threefold. First, we present an optimization framework to explore the applicability of the shadow computing model to support fault-tolerance in a high-performance computing environment, where the main computation is expected to execute at maximum speed in order to harness the full potential of the computing infrastructure. The model assumes that the desired level of fault tolerance can be achieved with the instantiation of a single shadow process, although the model can be extended to support multiple shadow processes.

Second, using the optimization framework, we propose and study three implementation methods of the shadow computing model. The first method, referred to as "stretched" replication, takes a straight-forward approach to minimizing energy by computing a uniform speed of execution across the execution interval. Although simple to implement, stretched replication is oblivious to the dynamics of the failure, resulting in sub-optimal energy performance. The second method, referred to as minimum-work replication, takes into consideration the minimum time between failures for a single node is likely to exceed the execution time of the task. Based on this observation, the method computes a near-optimal execution speed of the shadow. The third method, referred to as optimal energy replication, addresses shortcomings in these two models and uses an optimal energy-consumption approach to derive the shadow's minimum-energy speed, both before and after failure, in order to meet the expected response time of the underlying application regardless of failure rates. It is worth noting that the implementation complexity of the optimal approach does not increase significantly in comparison with the stretched and minimum-work replication schemes.

Third, we propose a performance evaluation framework to assess the performance of the proposed methods for different workload characteristics and performance requirements of the main task. Throughout the study, the performance of the proposed shadow computing implementation methods are compared to that of the "pure" replication scheme described in [10]. The major difference between the methods proposed in this paper and current replication schemes is that shadow computing does not force the process replica to execute at the same speed as the main process. As the results of the performance study show, relaxing this restriction allows shadow computing to save up to 50% of the expected energy consumed by pure replication.

The remainder of the paper is organized as follows: Section 2 reviews work related to fault-tolerance in large-scale high-performance computing systems. Section 3 presents the shadow computing model and discusses the building components of its execution model. Section 4 the shadow computing model is cast as an energy optimization problem to achieve fault-tolerance, while minimizing energy consumption without violating the expected performance requirements of the supported application. In Is Section 5 three different approaches to solving the energy optimization problem and their potential implementation are explored. The methods differ in their approach to energy minimization and their reaction to failures. In Section 6, we develop a performance evaluation framework to analyze and assess the performance of the three shadow computing methods, including their sensitivity to critical workload and performance parameters. Throughout the study, the performance of the three methods is compared to pure replication, a recently proposed alternative to check-pointing in exascale computing. Section 7 presents the conclusion of this work and discusses future work.

2. RELATED WORK

Checkpointing and rollback recovery are frequently used to deal with failures in computing systems. The process involves periodically saving the current state of the computation in a stable storage, with the anticipation that in case of a system failure computation can be rolled back to the most recent safely saved state before failure occurred [1, 5, 6, 9, 8, 11, 14, 15, 17, 18].

Approaches to distributed checkpointing differ in the level of process coordination required to construct a consistent global state, the size of the state, and the frequency at which checkpointing occurs [20, 24]. Based on independent checkpointing, processes coordination among processes is not required, thereby considerably reducing checkpointing overhead. At the occurrence of a failure, all processes resume computation from the most recent consistent global state [3, 5, 13, 19].

The major drawback of this approach stems from the potentially high computational cost incurred to roll-back to a consistent global state, assuming that such a state exists. This process typically requires deep analysis of the dependencies among the distributed computations to determine the existence of a roll-back state or the need to resume computations from their initial states.

Coordinated checkpointing, a widely used technique to deal with failures in distributed systems, requires coordination among processes to establish a state-wide consistent state [5]. The major benefit of this approach stems from its simplicity and ease of implementation, which lead to its wide adoption in high-performance computing environments. However, its major drawback is lack of scalability, since it requires concurrent checkpointing. Approaches have been proposed to reduce the level of coordination depending on the nature of the applications and the patterns of communications among processes [13].

In communication-induced checkpointing schemes, processes perform independent checkpoints to provide the basis for a system-wide consistent state [2, 4]. Although it reduces the coordination overhead, the approach may lead processes to store useles states that are never used in future rollbacks. To address this shortcoming, "forced checkpoints" have been proposed [12]. The approach, in most cases, reduces the number of useless checkpoints; it may, however, lead to unpredictable checkpointing rates.

Despite numerous improvements of the basic checkpointing scheme, recent studies show that high failure rates in high-performance computing systems, coupled with high checkpointing and restart overhead, considerably reduces the practical feasibility of existing centralized, hard disk drive centric checkpointing and rollback recovery schemes. Additionally, checkpointing techniques produce a significant amount of energy overhead [7] and therefore exascale computing requires new approaches.

Recently, redundant computing has been proposed as a viable alternative to checkpointing to handle fault-tolerance requirements of high-performance applications [10, 21]. Process replication, however, can be prohibitively costly in terms of computing resources and energy consumption. To the best of our knowledge, this is the first attempt to explore a state-machine-replication based framework to reduce energy while meeting the computational requirements of the underlying application.

In addition to its significant energy saving, shadow computing has potential to significantly increase an application's mean time to interrupt (MTTI). This model can be further enhanced to allow for diversity through the execution of multiple shadow processes in different geographical environments, using different implementations of the underlying computation to potentially detect or correct faults that do not necessarily lead to abnormal termination of the process, but instead cause it to produce incorrect results.

3. MODEL AND DATA STRUCTURE

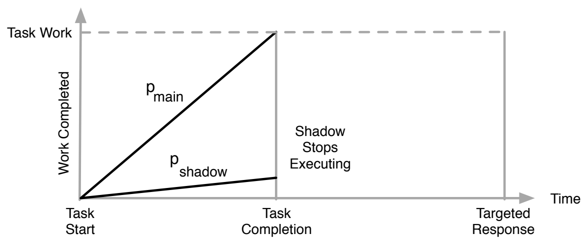
In this section, we provide an overview of the shadow computing execution model, under different failure scenarios. We also discuss the mapping of the processes to the computing infrastructure to ensure fault-tolerance to failure. Finally, we present the basic data structure that enables efficient communication between a main process and its associated shadow. The main purpose of this section is to show the feasibility of the shadow computing model. The details of how the execution model and its associated data structure are implemented is outside the scope of this work, and will be the subject of a future publication.

3.1 Execution Model

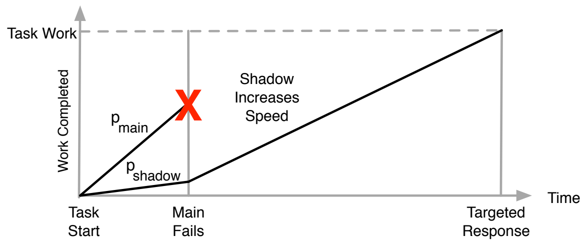
Depending on the occurrence of failure during execution, two scenarios are possible. The first scenario, depicted in Figure 3(a), takes place when no failure occurs[1]. In this scenario, the main process executes at the optimum processor speed, namely the speed necessary to achieve the desired level of fault-tolerance, minimize energy consumption and meet the target response time of the supported application. The figure shows the completion time of the task, in the absence of failure. During this time, the main process completes the total amount of work required by the underlying application. However, the shadow process, executing at a reduced processor speed, only completes a significantly smaller amount of the original workload. Because the likelihood of an individual node failure is low, this scenario is most likely to occur with high frequency, resulting in a relatively small amount of additional energy consumption to achieve fault-tolerance. The benefits of this scheme clearly outweigh the additional energy cost. Furthermore, it is worth noting that the failure of the shadow process does not impact the behavior of the main process.

The second scenario, depicted in Figure 3(b), takes place when failure of the main process occurs. Upon failure detection, the shadow process increases its processor speed and executes until completion of the task. The processor speed at which the shadow executes after failure is derived so that the shadow computing model guarantees that the task still completes by the targeted response time,regardless of when the failure occurs. Furthermore, shadow computing achieves

[1]For the purpose of this discussion, only a single shadow is considered. The discussion can be easily extended to deal with multiple shadow processes



(a) Shadow Computing Case of no failure



(b) Shadow Computing Case of failure

considerable energy saving by taking advantage of the fact that the likelihood of individual component failures in exascale computing is small, thereby making oblivious the need to execute "duplicate work" unless a failure occurs. It is assumed that shadow processes can detect failures although the details of this are beyond the scope of this paper.

It is worth noting that the interplay between resiliency and power management manifests itself in different ways and must be analyzed carefully. Operating at lower voltage thresholds, for example, reduces power consumption but has an adverse impact on the resiliency of the system to handling high error rates in a timely fashion. Our approach to deriving optimal execution speed for the main process and its associated shadows, both before and after failure, seeks to avoid continuous change in voltage and frequency to prevent potential thermal and mechanical stresses on the electronic chips and board-level electrical connections.

3.2 Process Mapping

In the shadow computing model the execution of a task spawns the creation of both a main process and a suite of shadow processes. These processes must be carefully mapped to the computing nodes of the exascale computing infrastructure to achieve fault-tolerant execution. Consequently, the mapping must be done such that the main and shadow processes are *fault-isolated* from each other, meaning that a fault affecting one process does not affect the other. Fault-isolation is necessary to minimize the likelihood that both the main and shadow processes fail at the same time. In high-performance computing, fault-isolation uses the multicore capabilities of the infrastructure to assign main processes and shadows in a way such a given shadow process can only be run along side an unrelated main process. Figure 3 illustrates a feasible assignment that satisfies such a constraint, for the case of three main processes and their associated shadows.

3.3 Message Passing

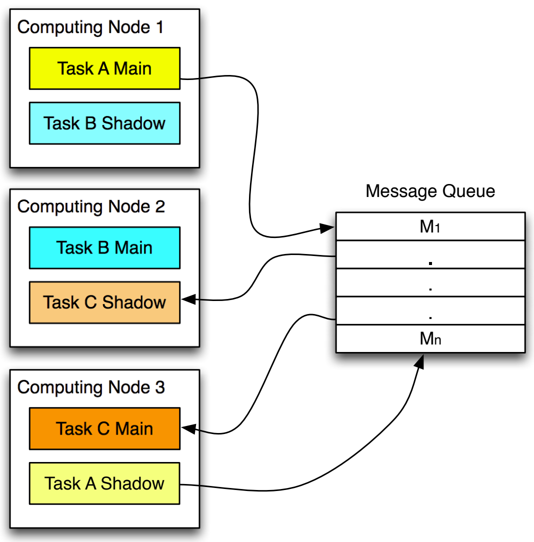


Figure 3: Example Process Mapping

Another important aspect of the shadow computing model is providing process communication to achieve synchronization and maintain system consistency. A communication model to ensure these requirements must at a minimum support these two properties:

- All messages destined for a task must be delivered to both the main process and all associated shadow processes.
- Any message previously sent from a task must not be duplicated by any of the running process.

To satisfy the communication and synchronization requirements the shadow computing model, the runtime support environment uses a Global Message Queue, see Figure 3. All inter-task communication will occur through a virtual message queue, which is assumed to be resilient to system faults. When a task spawns the main and shadow processes the queue is notified of all processes created. For scalability reasons we assume the queue is a *passive-queue*, meaning it only stores messages and waits for them to be requested as opposed to forwarding messages to processes. This eliminates the need for the queue to notify processes directly and instead lets the processes request them when they are ready. This allows processes running at a higher execution speed to not interfere with the execution of processes running slower.

When a message arrives at the queue for delivery to a task it will hold that message until it has been delivered to all associated processes[2]. This is possible because all associated processes were registered with the queue when created. An example of message delivery is depicted in Figure 4. While not depicted, messages would also also be removed from the queue once the task was completed. This scheme ensures that all executing processes will receive all messages destined for the task.

[2]Any implementation of such a system will have to address the issue of growing queue size but for this discussion it is assumed we have an infinite queue.

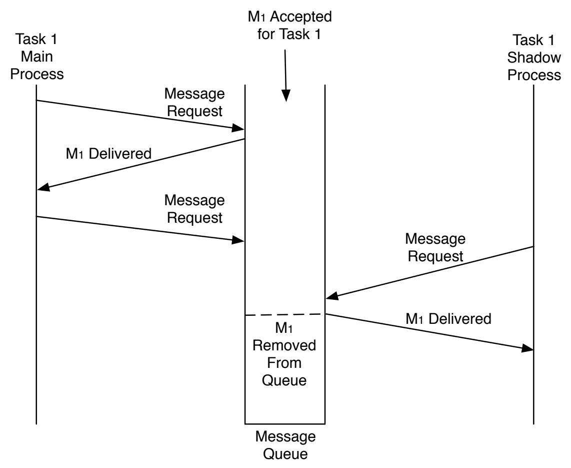


Figure 4: Example Message Delivery

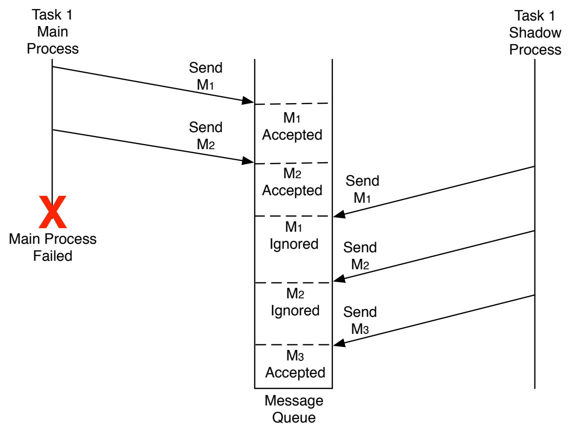


Figure 5: Example Message Receiving

In order to ensure that messages are not duplicated by shadow processes we propose that all messages be assigned a unique sequence number per task. When the queue receives a message from a task it will determine if that message has already been received by the queue for the task. If the message is a duplicate it will simply ignore the message, if however it is a new message it will queued for delivery. We show an example of the message receiving process in Figure 5. This allows shadow processes to execute slower and not produce duplicate system messages. The other benefit of this model is that messages will be processed regardless of their source, therefore the queue doesn't need to be aware of process failures.

4. ENERGY OPTIMIZATION MODEL

As stated previously, the basic idea of shadow computing is to associate a number of "shadow processes" with each main process. The main responsibility of a shadow process is to take over the responsibility of a failed main process and bring the computation to a successful completion. In this section, we define a framework for evaluating shadow computing and then use this to derive a model for representing the expected energy consumed by the system. We then describe in terms of this model three different methods for applying shadow computing in a high performance computing environment.

4.1 Shadow Computing Framework

We consider a distributed computing environment executing an application carried out by a large number of collaborative tasks. The successful execution of the application depends on the successful completion of all of these tasks. Therefore the failure of a single process delays the entire application, increasing the need for fault tolerance. Each task must complete a specified amount of work, W, by a targeted response time, R. The amount of work is expressed in terms of the number of cycles required to complete the task. Each computing node has a variable speed, σ, given in cycles per second and bounded such that $0 \leq \sigma \leq \sigma_{max}$. Therefore the minimum response time for a given task is $R_{min} = \sigma_{max} * W$.

In order to achieve our desired fault tolerance a shadow process executes in parallel with the main process on a different computing node. The main process executes at a single execution speed denoted as σ_m. In contrast the shadow process executes at two different speeds, a speed before failure detection, σ_b, and a speed after failure detection, σ_a. This is depicted in Figure 6.

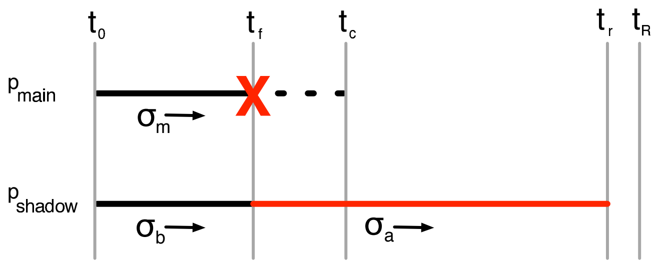


Figure 6: Overview of Shadow Computing

Based upon this framework we define some specific time points signaling system events. The time at which the main process completes a task, t_c, is given as $t_c = W/\sigma_m$. The time at which the shadow process completes as task, t_r, is given as $t_r = (W - \sigma_b t_c)/\sigma_a$ related but not necessarily equal is t_R which is the time the system reaches the targeted response time for a given task. Additionally, we define the time point t_f as the time at which a failure in the main process is detected.

Using this framework we formalize our objective as the following minimization problem.

$$\begin{aligned}
\text{minimize } & E(\sigma_m, t_0, t_f, t_c) + E(\sigma_b, t_0, t_f, t_c) + E(\sigma_a, t_f, t_r) \\
\text{subject to } & t_c \leq t_R \\
& t_r \leq t_R \\
& \sigma_m t_c \geq W \\
& \sigma_b t_c + \sigma_a (t_R - t_f) \geq W
\end{aligned} \tag{1}$$

Here the function $E(\sigma, t_0, t_1, t_2)$ represents the energy consumed by a process running at speed σ during the time period t_0 and $min(t_1, t_2)$. The first two constraints state that both the main process and the shadow process must complete by the targeted response time. The last constraints ensure that the amount of work done by those processes must be greater than or equal to the amount of work defined by the task.

It should be noted that it is assumed that node failures and task properties are unchangeable system properties therefore the system parameters we can change are the execution speeds of the processes. Thus the output of this optimization problem is the execution speeds, σ_m, σ_b and σ_a. In the proceeding sections we will present a power and failure model for individual computing nodes then use these to model the expected energy of the shadow computing system. Then in Section 5 we then apply this model to the high performance computing environment.

4.2 Power Model

It is well known that by varying the execution speed of the computing nodes one can reduce their power consumption at least quadratically by reducing their execution speed linearly. The power consumption of a computing node executing at speed σ is given by the function $p(\sigma)$, represented by a polynomial of at least second degree, $p(\sigma) = \sigma^n$ where $n \geq 2$. The energy consumed by a computing node executing at speed σ during an interval of length T is given by $E(\sigma, T) = \int_{t=0}^{T} p(\sigma) dt$. Throughout this paper we substitute the energy for a particular time interval, t, with the derived value $p(\sigma)t$, because $p(\sigma)$ is treated as a constant with respect to time. We further assume that the computing node speed is bounded by the following equation $0 \leq \sigma \leq \sigma_{max}$.

4.3 Failure Model

The failure can occur at any point during the execution of the main task and the completed work is unrecoverable. Because the processes are executing on different computing nodes we assume failures are independent events. We also assume that only a single failure can occur during the execution of a task. If the main task fails it is therefore implied that the shadow will complete without failure. We can make this assumption because we know the failure of any one node is a rare event thus the failure of any two specific nodes is very unlikely. In order to achieve higher resiliency one would make use of multiple shadow processes and this failure model will still be valid.

We assume that a probability density function, $f(t)$ ($\int_0^\infty f(t)dt = 1$), exists which expresses the probability of the main task failing at time t. It is worth noting, that the model does not depend on any specific distribution.

4.4 Energy Model

Given the power model and the failure distribution, the expected energy consumed by a shadow computing task can be derived. We start by considering the expected energy consumed by the main process and derive the following equation:

$$\int_{t=0}^{t_c} E(\sigma_m, t) f(t) dt + (1 - \int_{t=0}^{t_c} f(t) dt) E(\sigma_m, t_c) \tag{2}$$

This first term of the equation represents the expected amount of energy consumed by the main process if a failure occurs, while the second term represents the expected energy consumed if no failure occurs.

Similarly, we can calculate the expected energy consumed by the shadow process, as follows:

$$\begin{aligned} &\int_{t=0}^{t_c} E(\sigma_b, t) f(t) dt \\ &+ \int_{t=0}^{t_c} E(\sigma_a, (t_r - t)) f(t) dt \\ &+ (1 - \int_{t=0}^{t_c} f(t)dt) E(\sigma_b, t_c) \end{aligned} \tag{3}$$

The first term represents the expected energy consumed by the shadow executing at σ_b up until the main process fails. The middle term represents the expected energy consumed when the main process fails and the shadow begins to execute at the speed, σ_a. The last term is the expected energy consumed in the event that no failure occurs and the shadow executes at σ_b the entire duration of the main process.

The total energy consumed by a shadow computing task is the summation of the energy consumed by the main process and shadow process. To expand this model to represent multiple shadows we would multiple the energy consumed by the shadow by the total number of shadow processes. Given one shadow process we can combine equations (2) and (3) to produce the single model representing the total expected energy consumed.

$$\begin{aligned} &\int_{t=0}^{t_c} (E(\sigma_m, t) + E(\sigma_b, t)) f(t) dt \\ &+ \int_{t=0}^{t_c} E(\sigma_a, (t_r - t)) f(t) dt \\ &+ (1 - \int_{t=0}^{t_c} f(t)dt)(E(\sigma_m, t_c) + E(\sigma_b, t_c)) \end{aligned} \tag{4}$$

5. APPLICATION TO HPC

One of the primary goals of high performance computing is to achieve the maximum possible throughput of the system. Thus when we apply shadow computing to this environment we assume that the execution speed of the main process should be the maximum possible execution speed, $\sigma_m = \sigma_{max}$. If no failure occurs then the task will be completed as soon as possible, known as the minimum response time. If the main process fails it is assumed that the task has some laxity as to when it will complete. The amount of laxity is bounded by the task's targeted response time, which is the time at which the task must be completed regardless of failure. The targeted response time is typically represented as a laxity factor, α, of the minimum response time. For example if the minimum response time is 100 seconds and the targeted response time is 125 seconds, the laxity factor is 1.25.

We propose three different schemes for for applying shadow computing to high performance computing.

- Energy Optimal Replication - Shadow execution speeds are those that minimize the consumed energy and guarantee completion by the targeted response time. This requires us to find σ_b and σ_a that minimize equation 4.
- Stretched Replication - Shadow execution is set to a single speed that guarantees completion by the targeted response time, $\sigma_b = \sigma_a = W/R$.
- Minimum Work Replication - Shadow execution speed before failure, σ_b, is set to the minimum execution speed that enables the shadow to still met the targeted response time. We will show that this method is typically energy optimal.

The remainder of this section presents a solution to finding σ_b and σ_a for energy optimal replication. We start by specifying our failure and power model then derive a closed form solution that produces these execution speeds.

5.1 Failure Probability

In our energy model we assume failure is described using any probability density function. In the remaining sections we use the exponential probability density function because it is widely accepted as a model representative of independent node failures, therefore $f(t) = e^{-\lambda t}$. This distribution also has the benefit of being differentiable allowing us to produce a closed form equation.

5.2 Energy Function

In the remainder of this paper we need to select values for our power model. In section 4.2 we defined the energy function, $E(\sigma, T)$, as the integral of the power function over the interval T. In the remaining sections we assume that the power function is defined as the squared value of the speed.

$$P(\sigma) = \sigma^2 \tag{5}$$

Thus the energy function is defined as the following:

$$E(\sigma, T) = \int_{t=0}^{T} \sigma^2 dt = \sigma^2 t \tag{6}$$

5.3 Optimal Execution Speeds

Once we have a known failure and power model we can begin to solve the optimization problem. We first make the observation that the speed of the shadow after failure, σ_a, is dependent upon the the shadow speed before failure, σ_b, and the time of failure, t_f. It can trivially be shown that to conserve the most energy one would let σ_a be the slowest possible speed to finish by the targeted response time, R. Therefore σ_a is no longer constant with respect to the time of failure. From this observation the following value of σ_a can be derived.

$$\sigma_a = (W - \sigma_b * t_f)/(R - t_f) \tag{7}$$

Substituting this value into the energy model allows us to reduce the number of variables in our objective function, specifically this reduces the output of our optimization to one variable, σ_b.

The one constraint we have not considered in our optimization is that if the main process fails the shadow process must be able to complete the given work, W, by the targeted response time, R. This is known as the "work constraint" and is represented by the following inequality.

$$t_c * \sigma_b + (R - t_c) * \sigma_{max} \geq W \tag{8}$$

The intuition for this constraint is that in the worst case the shadow will have to execute at the maximum possible speed after failure to achieve the targeted response time. This enforces the constraint such that if the main process fails at the very last time point, t_c, then the shadow process will still be able to complete the work by the targeted response time. This places a lower bound on the value for σ_b and as we will later show typically determines the value of σ_b.

5.4 Optimal Shadow Computing - Solution

In the preceding sections we have specified components of our general energy model found in Equation 4. Our optimization problem is now well defined as finding a value for σ_b that minimizes the following function.

$$\begin{aligned}&\int_{t=0}^{t_c} (\sigma_{max}^2 t + \sigma_b^2 t) e^{-\lambda t} dt \\ &+ \int_{t=0}^{t_c} [(W - \sigma_b * t_f)/(R - t_f)]^2 (R - t) e^{-\lambda t} dt \\ &+ (1 - \int_{t=0}^{t_c} e^{-\lambda t} dt)(\sigma_{max}^2 t_c + \sigma_b^2 t_c)\end{aligned} \tag{9}$$

Also note that $t_c = W\sigma_{max}$ because the amount of work, W, is given and main process execution speed is defined as $\sigma_m = \sigma_{max}$. After solving equation 9 we can then take the derivative of the result with respect to σ_b and solve for σ_b.

$$\sigma_b = \frac{\begin{array}{l}(W(-E^{R\lambda} + E^{R\lambda+W\lambda} - E^{W\lambda}R\lambda Ei[R\lambda] + \\ E^{W\lambda}R\lambda Ei[(R-W)\lambda]))\end{array}}{\begin{array}{l}(E^{R\lambda}R - E^{R\lambda+W\lambda}r + E^{R\lambda}W + \\ E^{W\lambda}R^2\lambda Ei[R\lambda] - \\ E^{W\lambda}R^2\lambda Ei[(R-W)\lambda])\end{array}} \tag{10}$$

In this equation $Ei[x]$ represents the exponential integral function.

Equation 10 gives us the energy optimal execution speed for the shadow before a failure is detected, σ_b. However this execution speed does not guarantee that the shadow will be able to complete the work by the targeted response time, R. To ensure that the shadow completes enough work we impose the "work constraint" found in Equation 8. By solving the "work constraint" for σ_b we can use this as a lower bound; thus producing the final energy optimal execution speed for the shadow before failure, $\tilde{\sigma_b}$.

$$\tilde{\sigma_b} = Max[\sigma_b, (2W\sigma_{max} - R\sigma_{max}^2)/W] \tag{11}$$

Our final solution now outputs the shadows execution speed before failure, σ_b, given the following inputs:

- W - The amount of work necessary to complete the task.
- R - The targeted response time for the task.
- σ_{max} - The maximum possible execution speed.
- λ - Parameter of our probability density function used to model failure.

5.5 Minimum Work Replication

Most of the time the task size will be much smaller than the mean time between failure, MTBF, of individual computing nodes. Given this one can observe that the optimal solution will typically be bounded by the "work constraint". Therefore we propose that instead of calculating energy optimal execution speed one can achieve near optimal performance by letting the execution speed be derived by the constraint directly. Minimum work replication can be thought of as a simplification of the optimal solution derived in the previous section. This simplification is only feasible because we have fixed the execution speed of the main process.

$$\hat{\sigma_b} = (2W\sigma_{max} - R\sigma_{max}^2)/W \tag{12}$$

6. ANALYSIS

Using the energy model and solutions described in the previous sections we now compare the energy consumed by optimal energy shadow, stretched replication and minimum work replication. Without loss of generality, we assume σ_{max} is normalized such that $\sigma_{max} = 1$. Execution speeds will be presented as factors of that maximum speed. In addition to comparing energy savings we also consider the energy models sensitivity to input variables.

6.1 Energy Savings

The first and obvious analysis is to compare the expected energy consumption of shadow computing with pure replication. We represent "Energy Savings" as a percentage of the amount of energy saved when using shadow computing versus pure replication. This analysis makes use of the energy model described in Section 4 to calculate the total expected energy of a given task, which is assumed to have a main process and one shadow process. To calculate the energy consumed by pure replication we let $\sigma_m = \sigma_b = \sigma_a = \sigma_{max}$. When evaluating our methods we let the execution speeds be determined by the equations derived in Section 5.

6.1.1 Optimal Shadow

In figure 7 we show the energy savings for a 12 hour task for various targeted response time factors, α. Recall that α is the laxity we allow our tasks if there is failure in the main process. Note that if we let $\alpha = 1.0$ this is equivalent to pure replication and shadow computing provides no energy savings. This comparison looks at the savings as a function of the mean time between failure, MTBF, of individual computing nodes. We vary MTBF from 1/2 day to 25 days, this is short given todays component failure rates, however as the MTBF increases the amount of energy savings continue asymptotically to the values presented at 25 days.

The first observation is that if we allow the task to take twice as long upon failure, $\alpha = 2.0$, we can approach the maximum possible savings of 50%. Even if we have very little laxity, $\alpha = 1.25$, optimal shadow still saves upto 21% compared to pure replication. The maximum of 50% savings is because

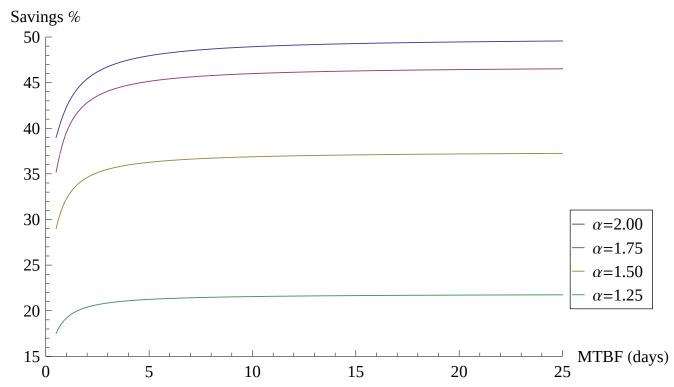


Figure 7: Energy savings for optimal shadow vs pure replication for a 12 hour task.

if the shadow process is given twice as much time to finish then optimal replication does little or no work until a failure occurs. The shadow will then execute at the maximum speed to achieve the targeted response time. If the likelihood of failure is high, low MTBF, then the optimal shadow will do more work prior to failure therefore expending more energy. You can see this effect on σ_b in Table 1. Observing at the bottom right corner you will notice that the shadow does very little if the likelihood of failure is low and we have enough time to "re-execute" the entire task. As we increase the failure rate or decrease laxity the optimal energy shadow increases the amount of work done prior to failure.

α / MTFB	1.25	1.50	1.75	2.00
0.5	0.80	0.67	0.57	0.500
1.0	0.80	0.67	0.57	0.500
5.0	0.78	0.65	0.55	0.484
10.0	0.75	0.56	0.47	0.407
25.0	0.75	0.50	0.30	0.251
1yr	0.75	0.50	0.25	0.024
3yr	0.75	0.50	0.25	0.012
5yr	0.75	0.50	0.25	0.005

Table 1: Optimal energy values of σ_b, $\sigma_{max} = 1.0$.

6.1.2 Stretched Replication

Recall that stretched replication sets the shadow's to a constant execution speed which is the minimum speed necessary to complete the job by the targeted response time, $\sigma_b = \sigma_a = W/R$. Figure 8 compares this method to the savings achieved by using the energy optimal solution. Despite being naive it still provides significant energy savings, between 17%-37% depending on the system laxity, represented by α.

The important observation is that the energy savings of stretched replication nearly matches energy optimal replication. When MTBF is low, stretched replication matches the optimal performance. This is because the optimal solution drives the shadow to complete as much work as possible because failure is very likely to occur, thus producing the same execution speeds as stretched replication. As MTBF increases the optimal will begin to perform better

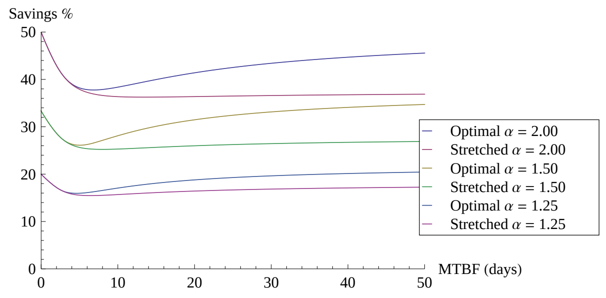


Figure 8: Energy savings for stretched and optimal replication. W=12 hours

than stretched replication, this is because the optimal solution will reduce the amount of work done before failure. This "gap" in energy savings reaches a stable distance as MTBF increases. Also, note that the decrease in α and the "gap" are correlated, demonstrated in Figure 9. The "gap" shrinks from 13% at $\alpha = 2.0$ to less than 1% at $\alpha = 1.10$.

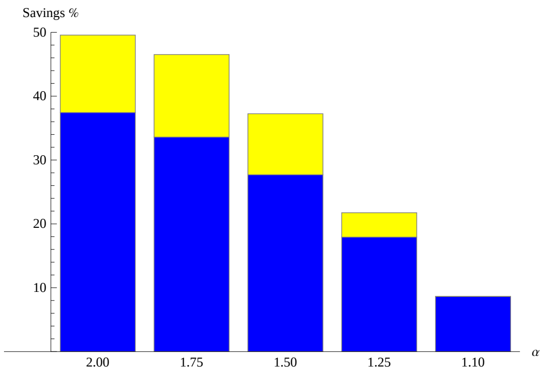


Figure 9: Energy savings "gap" between stretched and optimal. W=12 hours, MTBF=10days

6.1.3 Minimum Work Replication

As described in detail in Section 5.5 minimum-work replication is motivated by the assumption that the task size will typically be much less than the MTBF of individual nodes. When this is the case the optimal energy replication "hits" the work constraint because the likelihood of failure is low. Even when the task size is significantly larger than the individual nodes MTBF optimal replication is only slightly better than minimum work replication, see Figure 10.

6.2 Sensitivity to Alpha

The amount of energy savings achieved by either stretched or optimal shadow is highly dependent upon the amount of laxity the system has in the event of failure, represented by α. To see this effect we fix the MTBF and then observe the energy savings as a function of α, see Figure 11. The major observation here is that increasing α has largely a linear effect on the energy savings up until $\alpha = 2.0$ at which point the gain minimally increases to the maximum of 50% savings. While the job size does have an effect on actual energy savings, the trend is the same.

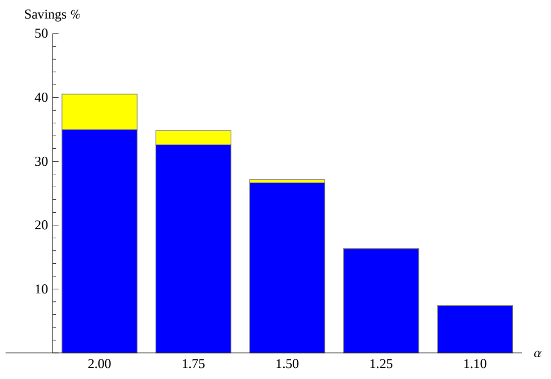


Figure 10: Energy savings "gap" between minimum-work and optimal replication. W=48 hours, MTBF=0.5days

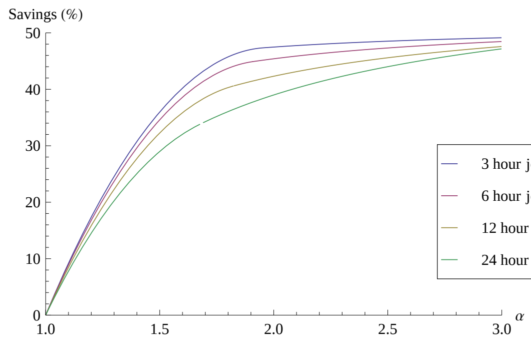


Figure 11: Energy savings for optimal shadow vs pure replication as a function of α. MTBF=1day

6.3 Sensitivity to Task Size

The task size has little effect on the trends observed; however, it does change the individual values and inflection points. Due to space reasons we omitted additional figures that demonstrate this however you can observe the behavior in Figure 11, which shows that the effect task size has on the energy savings is minimal as we vary α. Intuitively, there should be some impact because the longer a task runs the more likely it will be interrupted by a component failure. In Figure 12 we look at this in more detail by considering energy savings for a fixed MTBF and vary the task size. As expected when task size approaches the MTBF the energy savings also decreases, due to the optimal replication adjusting to account for the increased likelihood of task interruption by increasing the speed of the shadow before failure. Once the task size exceeds the MTBF, then the energy savings begin to increase again. If the main process is going to fail, it will do so more quickly and therefore will consume less energy. Our current model only assumes one node failure which is why the energy savings continues to increase beyond the MTBF, we discuss this as a potential future work in our conclusion.

7. CONCLUSION

The major contribution of this paper is a new technique, called "shadow computing" that provides energy-aware fault tolerance in large-scale distributed computing environments.

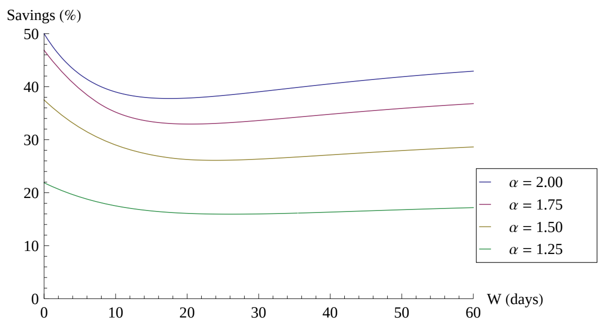


Figure 12: Energy savings for optimal shadow vs pure replication. MTBF = 10days

We also presented details on the shadow computing execution model and showed the feasibility of implementing such a system in distributed computing environment. We then explored different methods for applying this model to a high performance computing environment. Through this exploration we develop a general framework for evaluating the energy consumption of process replication schemes. Using this energy model we then analyzed our proposed solutions.

We proposed three different schemes for applying "shadow computing" to the high performance computing environment. Through evaluating the expected energy consumption of those schemes we showed that "shadow computing" has the ability to save significant energy. The energy savings is highly dependent upon the laxity available in the event of failure, represented as α in our analysis. We then showed that other factors such as component MTBF and task size have a minimum impact on the energy consumption. Furthermore, we demonstrated that simple, low-cost schemes such as stretched and minimum work replication can achieve near optimal behavior.

8. REFERENCES

[1] S. Agarwal, R. Garg, M. S. Gupta, and J. E. Moreira. Adaptive incremental checkpointing for massively parallel systems. In *Proceedings of the 18th annual international conference on Supercomputing*, ICS '04, pages 277–286, New York, NY, USA, 2004. ACM.

[2] L. Alvisi, E. Elnozahy, S. Rao, S. Husain, and A. de Mel. An analysis of communication induced checkpointing. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 242 –249, 1999.

[3] B. Bhargava and S.-R. Lian. Independent checkpointing and concurrent rollback for recovery in distributed systems-an optimistic approach. In *Reliable Distributed Systems, 1988. Proceedings., Seventh Symposium on*, pages 3 –12, oct 1988.

[4] D. Briatico, A. Ciuffoletti, and L. Simoncini. A Distributed Domino-Effect Free Recovery Algorithm. In *4th IEEE Symposyum on Reliability in Distributed Software and Database Systems*, 1984.

[5] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, Feb. 1985.

[6] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Gener. Comput. Syst.*, 22(3):303–312, Feb. 2006.

[7] M. el Mehdi Diouri, O. Gluck, L. Lefevre, and F. Cappello. Energy considerations in checkpointing and fault tolerance protocols. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1 –6, june 2012.

[8] E. Elnozahy and J. Plank. Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery. *Dependable and Secure Computing, IEEE Transactions on*, 1(2):97 – 108, april-june 2004.

[9] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, Sept. 2002.

[10] K. Ferreira, J. Stearley, J. H. Laros, III, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the viability of process replication reliability for exascale systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 44:1–44:12, New York, NY, USA, 2011. ACM.

[11] R. Geist, R. Reynolds, and J. Westall. Selection of a checkpoint interval in a critical-task environment. *Reliability, IEEE Transactions on*, 37(4):395 –400, oct 1988.

[12] J.-M. Helary, A. Mostefaoui, R. Netzer, and M. Raynal. Preventing useless checkpoints in distributed computations. In *Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on*, pages 183 –190, oct 1997.

[13] R. Koo and S. Toueg. Checkpointing and rollback-recovery for distributed systems. In *Proceedings of 1986 ACM Fall joint computer conference*, ACM '86, pages 1150–1158, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[14] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –9, april 2008.

[15] N. Naksinehaboon, Y. Liu, C. Leangsuksun, R. Nassar, M. Paun, and S. Scott. Reliability-aware approach: An incremental checkpoint/restart model in hpc environments. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 783 –788, may 2008.

[16] U. D. of Energy Office of Science. The opportunities and challenges of exascale computing, 2010.

[17] R. Oldfield, S. Arunagiri, P. Teller, S. Seelam, M. Varela, R. Riesen, and P. Roth. Modeling the impact of checkpoints on next-generation systems. In *Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on*, pages 30 –46, sept. 2007.

[18] A. J. Oliner, L. Rudolph, and R. K. Sahoo. Cooperative checkpointing: a robust approach to large-scale systems reliability. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 14–23, New York, NY, USA, 2006. ACM.

[19] K. Pattabiraman, C. Vick, and A. Wood. Modeling coordinated checkpointing for large-scale supercomputers. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, DSN '05, pages 812–821, Washington, DC, USA, 2005. IEEE Computer Society.

[20] J. Plank and M. Thomason. The average availability of parallel checkpointing systems and its importance in selecting runtime parameters. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 250 –257, 1999.

[21] R. Riesen, K. Ferreira, J. R. Stearley, R. Oldfield, J. H. L. III, K. T. Pedretti, and R. Brightwell. Redundant computing for exascale systems, December 2010.

[22] S. R. Sachs. Tools for exascale computing: Challenges and strategies, 2011.

[23] J. Srinivasan, S. Adve, P. Bose, and J. Rivers. The impact of technology scaling on lifetime reliability. In *Dependable Systems and Networks, 2004 International Conference on*, pages 177 – 186, june-1 july 2004.

[24] R. Strom and S. Yemini. Optimistic recovery in distributed systems. *ACM Trans. Comput. Syst.*, 3(3):204–226, Aug. 1985.