Noname manuscript No.
(will be inserted by the editor)

Lazy Shadowing

An Energy-Aware Resilience Approach for High Performance Computing

Xiaolong Cui · Bryan Mills · Taieb Znati · Rami Melhem

Received: date / Accepted: date

Abstract Current fault-tolerance approaches are designed to deal with both physical and logical fail stop errors and typically rely on the classic checkpoint-restart technique 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 infrastructures 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 Lazy Shadowing, a novel energy-aware computational model to support scalable fault-tolerance in future large-scale high-performance computing infrastructure. We present two schemes for applying lazy shadowing to high performance computing: energy optimal shadowing and stretched shadowing. All of the proposed schemes provide the same level of fault-tolerance as state machine replication while achieving significant energy savings.

Xiaolong Cui
University of Pittsburgh, U.S.
E-mail: mclarencui@cs.pitt.edu

Bryan Mills
University of Pittsburgh, U.S.
E-mail: bmills@cs.pitt.edu

Taieb Znati
University of Pittsburgh, U.S.
E-mail: znati@cs.pitt.edu

Rami Melhem
University of Pittsburgh, U.S.
E-mail: melhem@cs.pitt.edu

Keywords Fault Tolerance · Lazy Shadowing · Energy Awareness

1 Introduction

As our reliance on information technology continues to increase, the complexity and urgency of the problems our society will face in the future will increase much faster than are our abilities to understand and deal with them. It is expected that in the future, increasingly more complex applications will require very high computing performance and will process massive volume of data. Addressing these challenges requires radical changes in the way computing is delivered to support the computing requirements of these applications [25]. New algorithms and programming models must be developed to enable significantly higher levels of parallelism. It is expected that the number of concurrent threads to sustain these required levels of parallelism will rise to a billion, a factor of 10,000 greater than what current platforms can support. This in turn will result in a massive increase in the number of computing cores, memory modules and storage components.

A direct implication of these emerging trends is the need to address the difficult challenge of minimizing energy consumption. Furthermore, the fault rates are expected to dramatically increase, possibly by several orders of magnitude [26, 28]. Figure 1 shows the system Mean Time Between Failures (MTBF) and the number of faults as a function of the number of nodes in the system [24]. These faults, which can be transient, temporary, intermittent or permanent, stem from different causes and produce different effects. Concern about the increase of fault rates will not only be caused by the explosive growth in the number of computing and

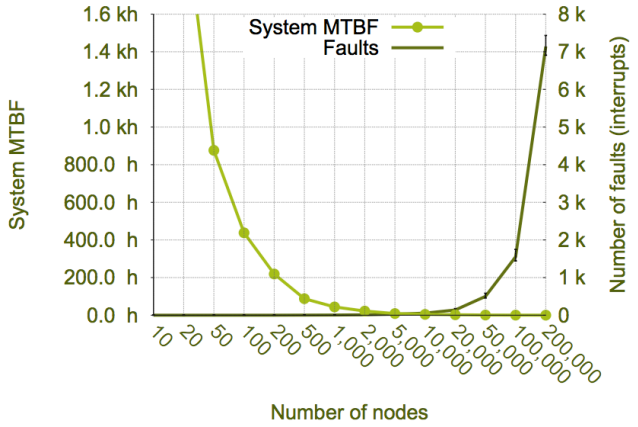


Fig. 1 Effect on system MTBF as number of nodes increase.

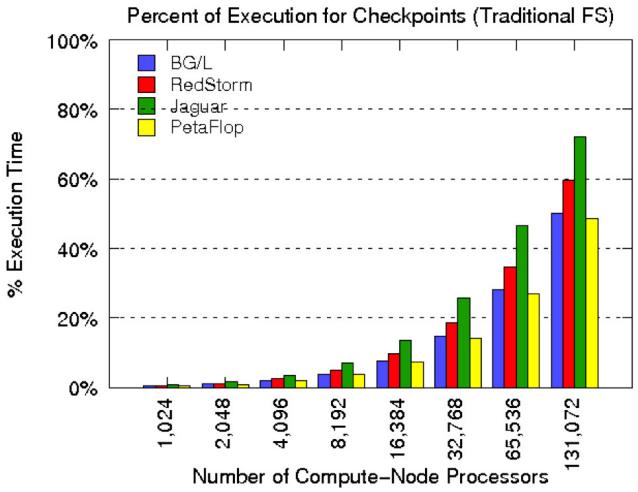


Fig. 2 Percent of time to perform checkpoints.

storage components, but will also grow out of the necessity to use advanced technology, operate at lower voltage levels, and deal with undesirable aging effects as they become significant [23]. Addressing this concern brings about unprecedented resilience challenges, which puts in question the ability of next generation HPC infrastructure to continue operation in the presence of faults without compromising the requirements of the supported applications.

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. 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.

In addition, current fault-tolerant approaches apply the same technique, mainly checkpoint-restart over the entire duration of the execution, to handle all types of faults, including permanent node crashes, transient computing errors, and IO device failures. The nature of errors in large-scale, high-performance computing environments, however, are such that a general and expensive fault-tolerance technique, such as checkpoint-restart, may not be an adequate approach to handle the diverse types of faults in these environments.

The main objective of this paper is to explore radically different paradigms to enable scalable resiliency with minimum energy consumption in future large-scale, high-performance computing environments. To this end, we propose a new energy-aware "lazy shadowing" model, which goes beyond traditional checkpointing and rollback recovery techniques and uses a multi-level, energy-aware replication approach to achieve scalable fault tolerance at scale.

The basic idea of the lazy shadowing 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 shadows are scheduled to execute concurrently with the main process, 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 within the targeted response time. Moreover, one among the remaining shadow processes becomes the primary shadow process.

In order to fully harness the potential of lazy shadowing to efficiently deal with failures, an optimization model is proposed to derive the execution rates of the main process and the prior- and post-failure execution rates of the shadow processes. The derived execution rates are such that the energy consumption is minimized, without violating the performance requirements of the underlying application. It is worth noting that the interplay between resilience and energy management manifests itself in different ways and must be analyzed carefully. Operating at lower voltage thresholds, for example, reduces power consumption but have adverse impact on the resilience of the system to handle high error rates in a timely and reliable fashion. Our approach will seek to avoid continuous change in voltage and frequency to prevent potential thermal and mechanical stresses on the electronic chips and board-level electrical connections.

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 lazy shadowing framework and formulates an energy optimization problem to achieve fault-tolerance, while minimizing energy consumption without violating the expected performance requirements of the supported application. In Section 4 two 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 5, we develop a performance evaluation framework to analyze and assess the performance of the three lazy shadowing methods, including their sensitivity to critical workload and performance parameters. Throughout the study, the performance of the two methods is compared to state machine replication, a recently proposed alternative to checkpointing in large-scale HPC. Section 6 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 [1,5,6,8,7,12,15,17–19].

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 [22,27]. 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 [3,5,14,20].

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 [14].

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 [13]. 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 [16].

Recently, redundant computing has been proposed as a viable alternative to checkpointing to handle fault-tolerance requirements of high-performance applications [9,24]. 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, lazy shadowing 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 Energy Optimization Model

As stated previously, the basic idea of lazy shadowing 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 lazy shadowing and then use this to derive a model for representing the expected energy consumed by the system.

3.1 Lazy Shadowing Framework

The focus of this paper is on non-interactive batch jobs, such as scientific applications, financial analytics and image processing running on a large-scale HPC infrastructure. This class of applications is typically defined in terms of their workload, maximum resource consumption, and completion time, which varies from several hours to several days. They are usually delay tolerant and can be run anytime, as long as the target deadline is met. Each application is composed of 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 $R_{min} = W/\sigma_{max}$.

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 3.

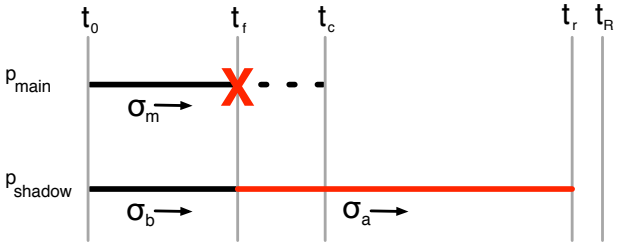


Fig. 3 Overview of lazy shadowing.

3.2 Power Model

Dynamic voltage and frequency scaling (DVFS) has been widely exploited as a technique to reduce CPU dynamic power [11,21]. It is well known that by varying the execution speed of the computing nodes one can reduce their dynamic 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 $p_d(\sigma)$, represented by a polynomial of at least second degree, $p_d(\sigma) = \sigma^n$ where $n \geq 2$. In addition to the dynamic power, CPU leakage and other components (memory, disk, network etc.) all contribute to static power consumption, which is constant as long as the system is on. In this paper we use ρ to represent the weight of static power and $1 - \rho$ the weight of dynamic power, when the node is running at the maximum speed. Therefore, the power consumption is expressed as $p(\sigma) = \rho \times \sigma_{max}^n + (1 - \rho) \times \sigma^n$. The energy consumed by a computing node executing at speed σ during an interval of length T is given by $E(\sigma, T) = \int_{t=0}^{T} p(\sigma)dt$. For an interval during which the speed is constant, the energy consumption can be calculated as $p(\sigma)t$. Throughout this paper we assume that dynamic power is cubic in relation to speed, i.e., $p(\sigma) = \rho \times \sigma_{max}^3 + (1 - \rho) \times \sigma^3$ We further assume that the computing node speed is bounded by the following equation $0 \leq \sigma \leq \sigma_{max}$.

3.3 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 lazy shadowing computation model does not depend on any specific distribution. However, in the remainder of this paper we use an exponential probability density function, $f(t) = \lambda e^{-\lambda t}$. Numerous works have studied the impact of DVFS on the failure rate or soft error rate [10,29]. To take this effect into consideration, we adopt the most widely used model from [29] and expressed it within our framework as $\lambda(\sigma) = \lambda_0 10^{1-\sigma/\sigma_{max}}$.

3.4 Energy Model

Given the power model and the failure distribution, the expected energy consumed by a lazy shadowing task can be derived as the weighted average considering all possible cases. First consider the case where no failure occurs and the main process successfully completes the task at time t_c^m.

$$E_1 = (1 - \int_0^{t_c^m} f_m(t)dt) \times (1 - \int_0^{t_c^m} f_s(t)dt) \times (E(\sigma_m, t_c^m) + E(\sigma_b, t_c^m)) \quad (1)$$

The first line is the probability of fault-free execution of the main process and shadow process. Then we multiple this probablity by the energy consumed by the main and the shadow process during this fault free execution, ending at t_c^m.

Next, consider the case where the shadow process fails at some point before the main process successfully completes the task.

$$E_2 = (1 - \int_0^{t_c^m} f_m(t)dt) \times \int_0^{t_c^m} (E(\sigma_m, t_c^m) + E(\sigma_b, t)) \times f_s(t)dt \quad (2)$$

The first factor is the probability that the main process does not fail, and the probability of shadow fails is included in the second factor which also contains the energy consumption since it depends on the shadow failure time. Energy consumption comes from the main process until the completion of the task, and the shadow process before its failure.

The one remaining case to consider is when the main process fails and the shadow process must continue to process until the task completes.

$$E_3 = (1 - \int_0^{t_c^m} f_s(t)dt) \times \int_0^{t_c^m} (E(\sigma_m, t) + E(\sigma_b, t) + E(\sigma_a, t_c^s - t)) f_m(t)dt \quad (3)$$

Similarly, the first factor expresses the probability that the shadow process does not fail. In this case, the shadow process executes from the beginning to t_c^s when it completes the task. However, under our "at most one failure" assumption, the period during which shadow process may fail ends at t_c^m, since the only reason why shadow process is still in execution after t_c^m is that main process has already failed. There are three parts of energy consumption, including that of main process before main's failure, that of shadow process before main's failure, and that of shadow process after main's failure, all of which depend on the failure occurrence time.

By summing these three parts we will have the expected energy consumed by lazy shadowing for completing a task.

$$E[\text{energy}] = (E_1 + E_2 + E_3) \quad (4)$$

3.5 Optimization problem formulation

Using the models above we formulate our objective as the following optimization problem.

$$\begin{aligned} \min_{\sigma_m, \sigma_b, \sigma_a} \; & E[energy] \\ s.t. \; & 0 \le \sigma_m \le \sigma_{max} \\ & 0 \le \sigma_b \le \sigma_m \\ & 0 \le \sigma_a \le \sigma_{max} \\ & t_m \le t_R \\ & t_s \le t_R \end{aligned} \quad (5)$$

The first three constraints represent the physical limit on the execution speeds. The last two constraints guarantee that the task can be completed by the target response time both with and without failure.

It should be noted that it is assumed that node failures are unchangeable system properties and task properties are given, 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.

4 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 lazy shadowing 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 two different schemes for for applying lazy shadowing 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 solve Equation 5.
- Stretched Replication - Shadow execution is set to a single speed that guarantees completion by the targeted response time, $\sigma_b = \sigma_a = W/R$.

The remainder of this section discusses the solution to finding σ_b and σ_a for energy optimal replication.

The last constraint in Equation 5 requires that if the main process fails the shadow process must be able to complete the given work, W, by the targeted response time, R. The effect of this constraint on the execution speeds depends on the failure detection time t_f. Since σ_b would be set to slower to save energy, the larger t_f is, the less time is left for the shadow to catch up, and thus σ_a needs to be larger. To enforce this constraint for all possible values of t_f, we transform it to the following inequality:

$$t_c * \sigma_b + (R - t_c) * \sigma_{max} \geq W \tag{6}$$

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. With this modification, we can use the MATLAB Optimization Toolbox to solve Equation 5 and derive the optimal σ_m, σ_b, and σ_a.

5 Analysis

With the analytical models developed in the previous sections, now we compare the energy savings among different fault tolerance approaches, i.e., lazy shadowing, stretched replication, and state machine replication. Without loss of generality, we assume the maximal execution rate is normalized such that $\sigma_{max} = 1$. The derived optimal execution rates are presented as fractions of the maximal execution rate.

During the experiments, we identified several critical parameters in the analytical models, which represents the characteristics of the task, and the underlying system. These parameters include:

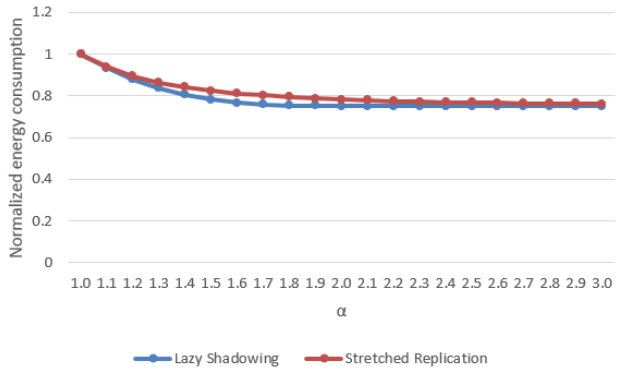
- ρ – static power ratio, which determines how much power consumption is independent of the execution rate.
- W – workload of the task.
- MTBF – the reliability of the system that runs the task.
- α – the laxity in the response time that can be tolerated.

In the following, we will present our sensitivity study results with respect to the above parameters. In each of the sensitivity study, we normalize the energy consumption of lazy shadowing and stretched replication to that of state machine replication.

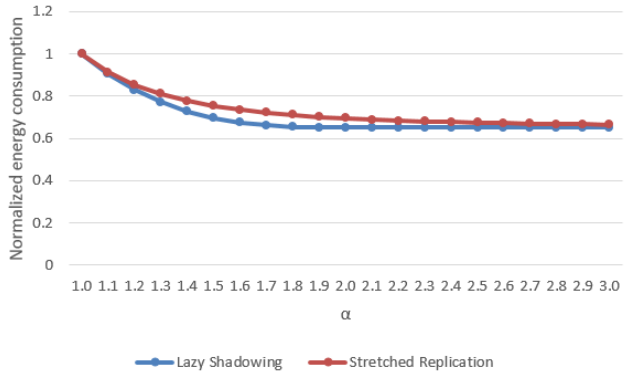
5.1 Response time laxity

Response time is always an important factor to consider in HPC as system efficiency is critical and high throughput is desirable. In the experiments we find that the energy savings of lazy shadowing and stretched replication versus traditional replication are largely impacted by the laxity in response time. This is mainly due to the fact that both of the proposed approaches rely on reducing the execution rates to save energy, while the laxity in response time determines how much the execution rate can be reduced. The results for $W = 240$ hours and MTBF=5 years are shown in Figure 4. Two values for ρ are used.

Figure 4 shows that both lazy shadowing and stretched replication can achive over 20% of energy savings compared to state machine repilcation, when there is enough laxity in the response time. When there is no laxity in the response time ($\alpha = 1$), lazy shadowing and stretched



(a) $\rho = 0.5$



(b) $\rho = 0.3$

Fig. 4 Normalized energy consumption for various response time laxities. W=240 hours, MTBF=5 years.

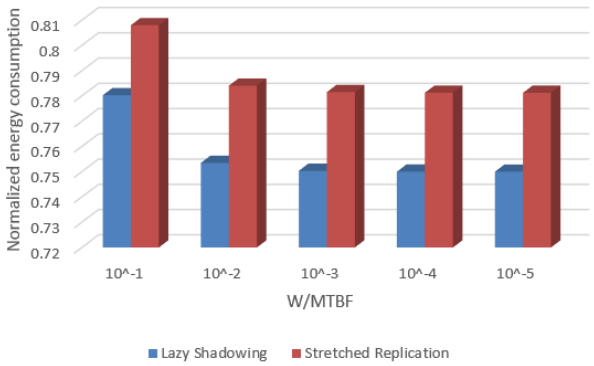


Fig. 5 Normalized energy consumption for various task vulnerabilities. ρ=0.5.

replication don't have any freedom to reduce the execution rate of the shadow process, thus will execute both the main and shadow processes at the maximal execution rate, which essentially converges to state machine replication. In this case, it is staightforward that all three approaches have the same energy consumption. However, as laxity in response time increases, lazy shadowing and stretched replication immediately gain the ability to slow down the shadow process and thus save energy. It is also clear that laxy shadowing has more potential than stretched replication in energy saving. It can take better advantage of the laxity in respone time as it is more flexible in controling the execution rate than stretched replication. The highest difference is 4.5% when α is around 1.5. As the laxity keeps increasing, the energy savings by lazy shadowing and stretched replication flatten out, as a result of the static power consumption. Since the static power consumption is independent of the execution rate, reducing the execution rate would extend the execution time, resulting in an increase in the energy consumption corresponding to the static power. Therefore, the minimal energy consumption would be achieved when there is a balance between energy from dynamic power and energy from static power, which may not be necessarily equivalent to using all the laxity in response time. It can be projected that if the staic power ratio is lower, laxy shadowing and stretched replication can take advantage of more laxity and achieve more energy saving compared to state machine replication. This will be discussed further in later section. At the right side of Figure 4(a) and Figure 4(b), lazy shadowing and stretched replication converge when there is enough laxity for both of them.

5.2 Task vulnerability

The objective of lazy shadowing is to minimize the expected energy consumption, considering the characteristics of both the task to execute and the underlying system. The probability of failure during the execution of the task is an important factor that will impact how the proposed fault tolerance approaches execute the task. Specifically, they will choose different process execution rates according to the likelyhood of failures. The ratio between the task workload and the MTBF is an approximation of the likelyhood of failure, assuming task workload is far less than the MTBF. In our experiments, we find that there is a linear relationship between task workload and MTBF in determining the optimal execution rates. Specifically, increasing the task workload has the same effect as decreasing the MTBF in our analytical models. Therefore, we combine them as a single parameter and refer to it as task vulnerability.

The results of the sensitivity study to task vulnerability, when $\rho = 0.5$, are shown in Figure 5. From left to right, the task vulnerability decreases, meaning that the likelyhood of failure decreases. We can see that for all task vulnerabilities considered, both lazy shadowing and stretched replication can achieve significant energy savings over state machine replication. This in-

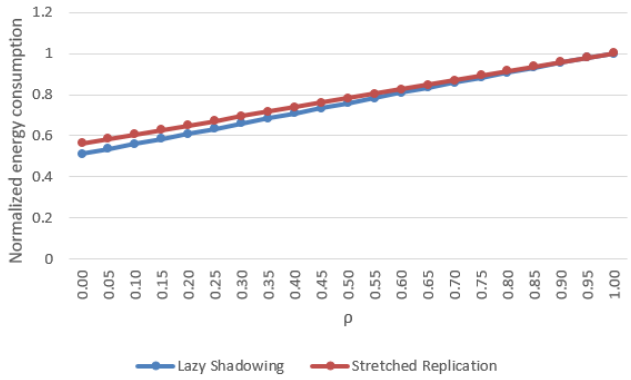


Fig. 6 Normalized energy consumption for various static power ratios. W=240 hours, α=2, MTBF=5 years.

dicates that the proposed approaches are scalable to future large-scale and failure-prone HPC environments. As task vulnerability decreases, lazy shadowing and stretched replication gains more energy savings. This is because the probability that the main process can complete the task without failure increases when task vulnerability decreases, increasing the probability that the shadow process can be terminated early to save energy. Again, lazy shadowing outperforms stretched replication because of its more flexibility in controling the execution rate of its shadow process.

5.3 Static power ratio

The computing nodes of each supercomputers have different power consumption characteristics, depending on the various computer organization and architecture technologies. Since our approaches are based on DVFS to control the process execution rates, the power consumption is partitioned into dynamic power, which has a superlinear relationship with the execution rate, and static power, which is unaffected by the execution rate. Static power ratio is an effecitve way to abstract the power saving ability of the proposed approaches on different machines.

DVFS saves dynamic power while not changing static power. These two aspects have conflicting effects on the total energy consumption. By reducing the execution rate, the execution time increases proportionally. This also increases the energy consumption from static power proportionally as the static power is constant. However, since the dynamic power can be reduced superlinearly, the energy consumption corresponding to dynamic power is reduced, even though the execution is extended. The above analysis can be illustrated with Figure 6. In addition, Figure 6 reveals that lazy shadowing can save up to 49% of energy over state machine replication when consider dynamic power only. On the other hand, lazy shadowing and stretched replication are forced to execute both the main and shadow processes at the maximal rate when dynamic power is negligible, converging to the behavior of state machine replication. Current supercomputers have a static power ratio between 40%-70%, and it is reasonable to suspect that this will continue to be the case. Within this range, the energy saving is 14%-29% for lazy shadowing, and 13%-26% for stretched replication.

6 Conclusion

The major contribution of this paper is a new technique, called "lazy shadowing" that provides energy-aware fault tolerance in large-scale distributed computing environments. We also presented details on the lazy shadowing 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 "lazy shadowing" 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.

References

1. Agarwal, S., Garg, R., Gupta, M.S., Moreira, J.E.: Adaptive incremental checkpointing for massively parallel systems. In: Proceedings of the 18th annual international conference on Supercomputing, ICS '04, pp. 277–286. ACM, New York, NY, USA (2004). DOI 10.1145/1006209.1006248. URL http://doi.acm.org/10.1145/1006209.1006248
2. Alvisi, L., Elnozahy, E., Rao, S., Husain, S., de Mel, A.: An analysis of communication induced checkpointing. In: Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on, pp. 242 –249 (1999). DOI 10.1109/FTCS.1999.781058

3. Bhargava, B., Lian, S.R.: Independent checkpointing and concurrent rollback for recovery in distributed systems-an optimistic approach. In: Reliable Distributed Systems, 1988. Proceedings., Seventh Symposium on, pp. 3 –12 (1988). DOI 10.1109/RELDIS.1988.25775
4. Briatico, D., Ciuffoletti, A., Simoncini, L.: A Distributed Domino-Effect Free Recovery Algorithm. In: 4th IEEE Symposyum on Reliability in Distributed Software and Database Systems (1984)
5. Chandy, K.M., Lamport, L.: Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Syst. **3**(1), 63–75 (1985). DOI 10.1145/214451.214456. URL http://doi.acm.org/10.1145/214451.214456
6. Daly, J.T.: A higher order estimate of the optimum checkpoint interval for restart dumps. Future Gener. Comput. Syst. **22**(3), 303–312 (2006). DOI 10.1016/j.future.2004.11.016. URL http://dx.doi.org/10.1016/j.future.2004.11.016
7. Elnozahy, E., Plank, J.: 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 (2004). DOI 10.1109/TDSC.2004.15
8. Elnozahy, E.N.M., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. **34**(3), 375–408 (2002). DOI 10.1145/568522.568525. URL http://doi.acm.org/10.1145/568522.568525
9. Ferreira, K., Stearley, J., Laros III, J.H., Oldfield, R., Pedretti, K., Brightwell, R., Riesen, R., Bridges, P.G., Arnold, D.: 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, pp. 44:1–44:12. ACM, New York, NY, USA (2011). DOI 10.1145/2063384.2063443. URL http://doi.acm.org/10.1145/2063384.2063443
10. Firouzi, F., Salehi, M.E., Wang, F., Fakhraie, S.M., Safari, S.: Reliability-aware dynamic voltage and frequency scaling. In: VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on, pp. 304–309. IEEE (2010)
11. Flautner, K., Reinhardt, S., Mudge, T.: Automatic performance setting for dynamic voltage scaling. Wirel. Netw. **8**(5), 507–520 (2002). DOI 10.1023/A:1016546330128. URL http://dx.doi.org/10.1023/A:1016546330128
12. Geist, R., Reynolds, R., Westall, J.: Selection of a checkpoint interval in a critical-task environment. Reliability, IEEE Transactions on **37**(4), 395 –400 (1988). DOI 10.1109/24.9847
13. Helary, J.M., Mostefaoui, A., Netzer, R., Raynal, M.: Preventing useless checkpoints in distributed computations. In: Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on, pp. 183 –190 (1997). DOI 10.1109/RELDIS.1997.632814
14. Koo, R., Toueg, S.: Checkpointing and rollback-recovery for distributed systems. In: Proceedings of 1986 ACM Fall joint computer conference, ACM '86, pp. 1150–1158. IEEE Computer Society Press, Los Alamitos, CA, USA (1986). URL http://dl.acm.org/citation.cfm?id=324493.325074
15. Liu, Y., Nassar, R., Leangsuksun, C., Naksinehaboon, N., Paun, M., Scott, S.: 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, pp. 1 –9 (2008). DOI 10.1109/IPDPS.2008.4536279
16. el Mehdi Diouri, M., Gluck, O., Lefevre, L., Cappello, F.: Energy considerations in checkpointing and fault tolerance protocols. In: Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on, pp. 1 –6 (2012). DOI 10.1109/DSNW.2012.6264670
17. Naksinehaboon, N., Liu, Y., Leangsuksun, C., Nassar, R., Paun, M., Scott, S.: 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, pp. 783 –788 (2008). DOI 10.1109/CCGRID.2008.109
18. Oldfield, R., Arunagiri, S., Teller, P., Seelam, S., Varela, M., Riesen, R., Roth, P.: Modeling the impact of checkpoints on next-generation systems. In: Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on, pp. 30 –46 (2007). DOI 10.1109/MSST.2007.4367962
19. Oliner, A.J., Rudolph, L., Sahoo, R.K.: Cooperative checkpointing: a robust approach to large-scale systems reliability. In: Proceedings of the 20th annual international conference on Supercomputing, ICS '06, pp. 14–23. ACM, New York, NY, USA (2006). DOI 10.1145/1183401.1183406. URL http://doi.acm.org/10.1145/1183401.1183406
20. Pattabiraman, K., Vick, C., Wood, A.: Modeling coordinated checkpointing for large-scale supercomputers. In: Proceedings of the 2005 International Conference on Dependable Systems and Networks, DSN '05, pp. 812–821. IEEE Computer Society, Washington, DC, USA (2005). DOI 10.1109/DSN.2005.67. URL http://dx.doi.org/10.1109/DSN.2005.67
21. Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proc. of the 18^{th} ACM Symp. on Operating systems principles, SOSP, pp. 89–102 (2001). DOI 10.1145/502034.502044. URL http://doi.acm.org/10.1145/502034.502044
22. Plank, J., Thomason, M.: 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, pp. 250 –257 (1999). DOI 10.1109/FTCS.1999.781059
23. Plank, J.S., Xu, J., Netzer, R.H.: Compressed differences: An algorithm for fast incremental checkpointing. University of Tennessee, Tech. Rep. CS-95-302 (1995)
24. Riesen, R., Ferreira, K., Stearley, J.R., Oldfield, R., III, J.H.L., Pedretti, K.T., Brightwell, R.: Redundant computing for exascale systems (2010)
25. Sachs, S.R.: Tools for exascale computing: Challenges and strategies (2011)
26. Srinivasan, J., Adve, S., Bose, P., Rivers, J.: The impact of technology scaling on lifetime reliability. In: Dependable Systems and Networks, 2004 International Conference on, pp. 177 – 186 (2004). DOI 10.1109/DSN.2004.1311888
27. Strom, R., Yemini, S.: Optimistic recovery in distributed systems. ACM Trans. Comput. Syst. **3**(3), 204–226 (1985). DOI 10.1145/3959.3962. URL http://doi.acm.org/10.1145/3959.3962
28. Torrellas, J.: Architectures for extreme-scale computing. Computer (11), 28–35 (2009)
29. Zhu, D., Melhem, R., Mossé, D.: The effects of energy management on reliability in real-time embedded systems. In: Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on, pp. 35–40. IEEE (2004)