Energy Consumption of Resilience Mechanisms in Large Scale Systems

Bryan Mills, Taieb Znati and Rami Melhem
Department of Computer Science
University of Pittsburgh
{bmills|znati|melhem}@cs.pitt.edu

Kurt B. Ferreira and Ryan E. Grant
Scalable System Software
Sandia National Laboratories
{kbferre|regrant}@sandia.gov

***Index Terms*—shadow computing, fault tolerance, scheduling, resilience, energy-aware**

***Abstract*—As HPC systems continue to grow to meet the requirements of tomorrow's exascale-class systems, two of the biggest challenges are power consumption and system resilience. On current systems, the dominant resilience technique is checkpoint/restart. It is believed, however, that this technique alone will not scale to the level necessary to support future systems. Therefore, alternative methods have been suggested to augment checkpoint/restart – for example process replication. In this paper we address both resilience and power together, this is in contrast to much of the competed work which does so independently. Using an analytical model that accounts for both power consumption and failures, we study the performance of checkpoint and replication-based techniques on current and future systems and use power measurements from current systems to validate our findings. Lastly, in an attempt to optimize power consumption for replication, we introduce a new protocol termed *shadow replication* which not only reduces energy consumption but also produces faster response times than checkpoint/restart and traditional replication when operating under system power constraints.**

I. Introduction

The race to build the worlds first exascale-class system has been underway for the last 10 years and many challenges remain. Two of the biggest challenges facing these future systems are power and resilience, each a direct result of the massive amount of parallelism necessary to achieve this goal. Delivering exascale performance could require a system over a million sockets, each supporting many cores [1]. This would result in a system with many-millions of components including increases in memory modules, communication networks and storage devices. With this explosive growth in component count will come a sharp decrease in the overall system reliability and an increase in system power requirements.

System power is a leading design constraint on path to exascale, established by the DOE at no more than 20MW [1]. This challenges the research community to provide a 1000x improvement in performance with only a 10x increase in power. It is expected that exascale-class machines will be capable of consuming more power than that set by the power cap. For example a system might have 150,000 sockets each consuming 200 watts of power at full speed, therefore if all sockets were operating at full power we would be consuming 30 mega-watts. To stay under the 20 mega-watt limit we would need to power off 50,000 of these sockets, or reduce the power consumption of some or all of the cores to stay within budget. While this may seem inefficient, as more hardware is available than can be supported by the power infrastructure, not all applications will be capable of fully utilizing system.

Maintaining efficiency will also be a significant challenge due to the increasing number of expected faults. As the number of components grow, system failures will become routine. Therefore, any resilience scheme must consider its effect on the application's energy and power consumption. In today's systems the response to faults mainly consists of restarting the application, including those components of its software environment that have been affected by the fault. To avoid full re-execution, these techniques checkpoint the execution periodically. Upon the occurrence of a hardware or software failure, recovery is then achieved by restarting the computation from a known good checkpoint.

Given the anticipated increase in failure rate and the time required to checkpoint large-scale compute- and data-intensive applications, it is predicted that the time required to periodically checkpoint an application and restart its execution will approach the system's mean time between failures (MTBF) [5]. Consequently, applications will make little forward progress, thereby reducing considerably the overall performance of the system [13], [15].

To increase overall system performance process replication has been proposed as a scalable fault tolerance method that can be more efficient in exascale-class systems [5]. A major criticism of replication is the necessary additional resources, especially when considering the power limitations imposed in exascale-class machines.

The objective of this paper is to compare the power and energy consumption of coordinated checkpointing and replication techniques. By looking at fault tolerance from the perspective of power we have found opportunities for making power-aware optimizations to replication. To this end, we propose *shadow replication*, a power-aware process replication protocol which provides faster response times and is more

This work is supported in part by NSF grants CNS12-53218 and CNS12-52306

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U. S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

efficient than both checkpoint/restart and traditional replication. We show that *shadow replication* can save 40% of the consumed energy while also being 40% faster in exascale-class machines.[1]

The remainder of the paper is organized as follows: section II provides a brief review of work in this area. Section III provides a description of the resilience methods explored in this paper. Section IV introduces the analytical framework used to model the behaviors of these methods in large-scale systems, presenting this analysis in Section V. We validate our methods in current systems using experimental data presented in Section VI. We then provide some conclusing remarks in Section VII.

II. Related Work

The analysis of energy/power concern and resilience for HPC is still in its formative stages and as such we are only aware of two papers which look at the energy consumption of fault tolerance schemes. In [4], the authors measure energy consumption of the three main tasks associated with checkpoint-restart methods: writing the checkpoint, recovering from a checkpoint and message logging. They make no attempt to optimize these tasks nor do they explore replication techniques. In [12], the authors look at three different checkpointing techniques and evaluate the energy consumption required. They find that uncoordinated checkpointing with parallel recovery was the best technique at both small and large scale saving up to 17% at 256,000 sockets. Also they show that as the number of sockets grows beyond 256,000 the trend in energy savings of parallel recovery is decreasing. Our work shows that replication increases energy savings as the system size grows, in contrast to this fault tolerance technique.

While not focused upon energy consumption, there is much research attempting to expand checkpoint techniques to exascale-class machines. This work revolves around two main concepts, reducing the checkpoint time and enhancing uncoordinated checkpointing. To our knowledge there is no work looking at optimizing the energy/power consumption of replication to make it a more viable solution for exascale-class environments.

III. Resilience Methods

A. Coordinated Checkpoint/Restart

Coordinated checkpoint/restart methods are the most widely used fault tolerance method in HPC environments. The dominant reason for this popularity is its simplicity to implement and the natural synchronization points required are present in most applications. In coordinated checkpointing, all running processes periodically pause their execution and write their state to a stable storage device. Once they have finished writing, the application proceeds with its execution. In the event of a failure, all processes restore from the last good checkpoint and resume execution collectively from that point.

[1]Example shown in detail in Section V, assumes a socket MTBF of 25 years and 53,000 sockets.

B. Uncoordinated Checkpointing

Another approach that has been suggested to improve the performance of checkpointing systems is uncoordinated or asynchronous checkpointing [2], [8], [9]. In these systems, nodes generally checkpoint and restore from local storage without the synchronization used by coordinated checkpointing. To support a node restoring from a local asynchronous checkpoint, nodes in this approach keep a log of recent messages that they have sent. When a node restores from a previous checkpoint, it can then replay reception of messages using remote nodes' logs.

While this approach can increase checkpointing performance, it also generally assumes the availability of local storage. In addition, logging increases the latency of messaging operations and potentially takes significant amounts of space. Finally, asynchronous checkpointing approaches can result in cascading rollbacks; recent work attempts to bound the amount of rollback that may be necessary [7], but also places nontrivial limits on application behavior.

C. Traditional Replication

Traditional replication is a method in which each application process is replicated on independent computing nodes, such that if one process fails its replica process can continue executing as if the failure did not occur. This is also referred to as process replication and has long been deployed in mission critical applications. Replication in HPC has largely been dismissed because of the additional resources required, but in recent years has been revisited [5] because of the increased failure rates expected in exascale-class machines. Furthermore, in this paper we show that power-aware replication techniques can mitigate this concern by using fewer resources.

D. Replication Optimizations

In this work we propose and analyze the potential of a number of power-aware optimizations to traditional replication termed *stretched* and *shadow* replication.

1) Stretched Replication: Stretched replication works on the assumption that performing work slowly can save energy. This is typically done through the use of dynamic voltage and frequency scaling (DVFS); while this is widely available in modern HPC environments it is rarely used. Stretched replication is a naïve approach which slows down the execution of all processes to the slowest possible speed while maintaining the applications targeted response time.

There are at least two reasons one might want to reduce the execution speed of nodes in an HPC environment. The first we have already mentioned, reducing the execution speed might be necessary to satisfy power limits. Another reason is that coordinated checkpointing already significantly increases the application's time to solution due to the checkpoints and restarts. If we can increase reliability by slowing down while still staying below this checkpoint slowdown, we can maintain an efficiency greater than that of traditional checkpointing.

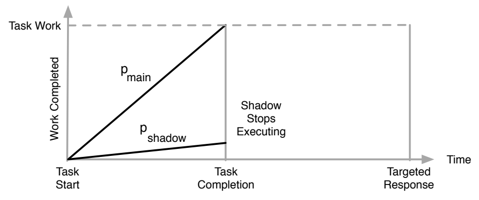
2) Shadow Replication: Stretched replication assumes that by going slower one can always save energy regardless of how long it actually takes to complete the job. This is not always the case in today's computers because machines require a base amount power to operate regardless of the processor speed, we refer to this as the machine's "overhead" power. Previous studies [6], [10] have shown this to be somewhere between 50-85% of the computing nodes' total power. Our experimental results show an overhead power of 60-67%. As the time to solution is increased it results in more energy consumption over the entire job, resulting in a time vs power tradeoff.

Exploring the energy consumption of replication led to the idea that the replica processes could execute at different execution speeds, while still guaranteeing a response time as good or better than that provided by checkpointing. For each process, shadow replication associates a suite of "shadow processes", whose size depends on the "criticality" and performance requirements of the underlying application. In order to overcome failure, the shadow executes concurrently with the main process, with the shadow and main processes on separate computing nodes. To minimize power, when multiple shadows of a single process exist, each replica shadow operates 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, the primary shadow process takes over the role of the main process and resumes computation, possibly at an increased speed, in order to complete the task at a targeted response time.

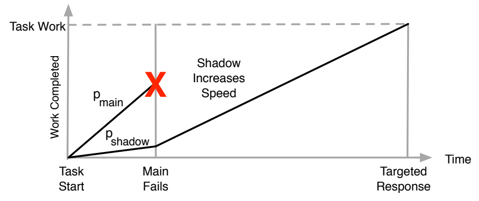
Depending on the occurrence of failure, two scenarios are possible. The first scenario, depicted in Figure 1(a), takes place when no failure occurs[2]. In this scenario, the main process executes at the optimum processor speed, namely the speed necessary to achieve the desired level of fault-tolerance, minimize power consumption and meet the target response time of the supported application. 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, completes a significantly smaller amount of the original work. Because the likelihood of an individual socket failure is low, this scenario is most likely to occur, resulting in a relatively small amount of additional energy consumption to achieve fault-tolerance.

The second scenario, depicted in Figure 1(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 determined such that the targeted response time is achieved. To maximize energy savings failure detection should occur as soon as possible but if this is not possible the shadow process could compute the time the main process should complete, then if that time is reached and the shadow has not been notified, it can assume a failure has occurred.

[2]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) Case of no failure



(b) Case of failure

Fig. 1. Shadow replication execution model.

As mentioned earlier, it is expected we will have additional sockets that will need to be powered off due to the power consumption limits. Given a power limit and socket power consumption there will be a fixed number of sockets available at any one time. Coordinated Checkpoint/restart would use all of the available sockets to perform work and in the event of failure rollback all sockets, therefore staying under the power limit by restricting the number of sockets used. Traditional replication would take half of the available sockets and use them as replicas, which has been shown to increase system efficiency in exascale-class machines. In contrast, shadow replication has the ability to use additional sockets because the replica sockets are consuming less power by running at a reduced speed. This has the added benefit of allowing additional sockets to work as main processes while still providing system resilience as in traditional replication. In the event of failure there is the potential delay in the time to solution because of the replica's slower execution speed. However, because of the ability to use additional sockets we can show that the expected time to solution is actually faster than both checkpoint/restart and traditional replication methods.

IV. Analytical Framework

To evaluate these methods on exascale-class machines we develop a framework of analytical models that represent the energy consumed for each of these methods. We first develop a model that describes the power consumption of a process or group of processes for a given type of work. Next, we derive a model that describes the expected amount of time those processes will be performing a given type of work. By combining these we can then estimate the total expended energy for a given application and system.

A. Computational Model

We consider a distributed computing environment of a large number of collaborative tasks (equivalent to ranks in MPI) which communicate frequently. The successful execution of the application depends on the successful completion of all tasks. Therefore, the failure of a single task delays the entire application. We assume the application is perfectly parallizable and has a total amount of work to accomplish, W. The work is assumed to be evenly divided into N tasks which also correspond to the number of system sockets. Because the work is evenly divided each socket will have $W_{task} = \frac{W}{N}$ work to complete. We assume an application with strong scaling, therefore as the number of sockets increases, the amount of work each individual process performs decreases. The amount of work is given in number of clock cycles and each computing socket has a variable speed, σ, given in clock cycles per second. Therefore the total solution time for an application when all sockets are operating at maximum speed is $T_s = \frac{W_{task}}{\sigma_{max}}$.

Each task also has an associated targeted response time, t_{resp}, which is the maximum time that the process will complete it's task. We will represent the targeted response time 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. In contrast, checkpointing techniques assume that if a failure occurs the system must always have enough time to re-execute. In our framework this results in $\alpha = 2.0$. [3]

B. Power Model

We start by considering the dynamic CPU power which is affected by the execution speed of the processor. Specifically, one can reduce the dynamic CPU power at least quadratically by reducing the execution speed linearly. Dynamic CPU power, P, can be determined by knowing the chip activity factor, A, the capacitance C, operating voltage, V, and the frequency, f. The dynamic CPU power is therefore represented by the function $P = A * C * V^2 * f$. In this paper we will be performing DVFS, resulting in a execution speed of σ. Because we are scaling both voltage and frequency, the last component allows us to represent power as the function $p(\sigma)$, as a polynomial of at least second degree, $p(\sigma) = \sigma^n$ where $n \geq 2$. In the remainder of this paper we assume that the dynamic power function is the cubic, $P(\sigma) = \sigma^3$.

Next, consider the "overhead" power which is consumed regardless of the speed of the processor. This includes both CPU static leakage and all other components consuming power during execution (memory, network, etc.). In this work we define overhead power to be a fixed factor, ρ, of the power consumed when the CPU is operating at full speed. The percentage of overhead power in a system is thus defined as $\frac{\rho}{\rho+1}$. By reducing the execution speed one can only change the dynamic power, the overhead power remains constant.

The energy consumed by a socket executing at speed σ during an interval $[t_1, t_2]$ is given by:

$$E_{soc}(\sigma, [t_1, t_2]) = \int_{t=t_1}^{t_2} (\sigma^3 + \rho\sigma_{max}^3)dt \tag{1}$$

[3]This assumes a single failure, if multiple failures occur checkpointing has the potential to have $\alpha > 2.0$.

The last component we consider is the energy consumed during an I/O operation, assumed to occur while writing or recovering a checkpoint. We handle this case separately because studies [4] have shown that this operation consumes a significant amount of energy. Similar to overhead power, we define maximum I/O power as a factor of the CPU when operating at full speed. This factor is defined as γ.

$$E_{io}([t_1, t_2]) = \int_{t=t_1}^{t_2} (\gamma\sigma_{max}^3)dt = (\gamma\sigma_{max}^3)(t_2 - t_1) \tag{2}$$

C. Failure Model

A failure can occur at any point during the execution of the main task and the completed work is unrecoverable. Because the tasks are executing on different computing nodes we assume failures are independent events and that only a single failure can occur during the execution of a task. We further assume that a probability density function, $f(t)$, exists which expresses the probability of the main task failing at time t. In the remainder of this paper we use this exponential probability density function, thus $f(t) = \frac{1}{M_{soc}} e^{-t/M_{soc}}$ where M_{soc} is the socket MTBF.

D. Checkpointing Energy Model

Coordinated checkpointing periodically pauses tasks and writes a checkpoint to stable storage. If any one socket fails then this checkpoint is read into memory and used to restart execution. Daly [3] computes the expected total wall clock time, t_w, given the original total solve time (T_s), a system MTBF (M_{sys}), checkpoint interval (τ), checkpoint time (δ) and recovery time (R). System MTBF is dependent upon the number of sockets and the socket MTBF, M_{soc}, this assumes that socket failures are independent events.

$$T_w = M_{sys} e^{R/M_{sys}} (e^{(\tau+\delta)/M_{sys}} - 1)(\frac{T_s}{\tau} - \frac{\delta}{\tau + \delta}) \tag{3}$$

Given this we can define the estimated energy required for a single process using checkpoint and restart (CPR) as E_{cpr}.

$$\begin{aligned} E_{cpr} =& E_{soc}(\sigma_{max}, [0, T_w]) \\ &+ E_{io}([0, \delta)]) \times \frac{T_s}{\tau} + E_{io}([0, R)]) \times \frac{T_w}{M_{sys}} \end{aligned} \tag{4}$$

The first part of this equation is because at any given time all processes are either working, writing a checkpoint or restoring from a checkpoint and all sockets are always executing at σ_{max}. The second part adds the energy required to write or restore from a checkpoint times the number of times we will be writing or recovering from a checkpoint. Lastly, we multiply this by the number of sockets, N.

E. Replication Energy Model

This section develops a model to represent the energy consumption of a replica pair. Then uses this model to determine the energy consumption of the combination of replication and checkpointing. This is equivalent to replacing a single socket in the checkpointing model described above with a replica pair.

Shadow replication has a main process executing at a single execution speed denoted as σ_m. One of the primary goals of high performance computing is to achieve maximum possible system throughput. Thus when we apply shadow replication 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. 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, depicted in Figure 2.

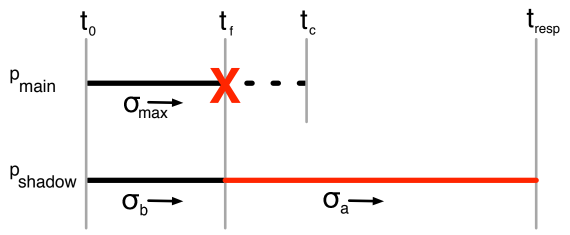


Fig. 2. Overview of Shadow Replication for HPC

We define some specific time points signaling system events. The time at which the main process completes a task, t_c, is given as $t_c = W_{task}/\sigma_{max}$. Additionally, we define t_f as the time at which a failure in the main process is detected. If no failure is detected we assume that $t_f = t_c$, this is done just to make the formulations below easier to understand. We can also define the time at which the shadow process will complete a task regardless of a failure as, $t_r = t_f + (W_{task} - \sigma_b t_f)/\sigma_a$.

We define the expected energy of a shadow replica-set as the summation of the expected energy consumed by the main and shadow process given our failure model. We assume at most one failure in the main process occurs but this can easily be extended to support multiple failures. In our analysis we have found that multiple failures made no discernible difference for socket MTBF's exceeding one year – a reliability easily reached for future systems.

$$\begin{aligned}E_{rep} = &\int_{t=0}^{t_c} (E_{soc}(\sigma_{max}, [0,t]) + E_{soc}(\sigma_b, [0,t]))f(t)dt \\ &+ \int_{t=0}^{t_c} E_{soc}(\sigma_a, [t, t_r])f(t)dt \\ &+ (1 - \int_{t=0}^{t_c} f(t)dt)(E_{soc}(\sigma_{max}, [0,t_c]) + E_{soc}(\sigma_b, [0,t_c]))\end{aligned} \tag{5}$$

The first part of this equation represents the expected energy consumed by the main and shadow process before a failure occurs in the main process. This is the summation of the expected energy consumed by the main plus the energy consumed by the shadow given our failure model over the total duration, 0 to t_c. The second part of this equation is the expected energy consumed by the shadow after failure occurs. The duration of this is from the time of failure, t_f, until the shadow completes execution, t_r. The last part is the expected energy consumed by the main and shadow processes in the event that no failure occurs.

This equation is then used as an objective function in the construction of an optimization problem used to find energy optimal execution speeds. We let the speed of the shadow process after failure be $\sigma_a = \sigma_{max}$, this is because we can trade the power consumed by the main process with the shadow process after failure of the main. This reduces the unknowns in the objective to the speed of the shadow process before failure, σ_b. Using traditional optimization techniques we take the derivative, set the result to zero and solve for σ_b. Additionally, we must constrain σ_b such that if the main process fails the shadow process will 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.

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

In addition to providing a model for shadow replication this can be used to represent traditional replication and stretched replication. Traditional replication would be represented by letting $\sigma_m = \sigma_b = \sigma_a = \sigma_{max}$. Stretched replication is represented by letting $\sigma_m = \sigma_b = \sigma_a = \frac{W_{task}}{t_{resp}}$.

V. Analysis

Our analysis finds several system parameters to be important in determining which fault tolerant method is most efficient.

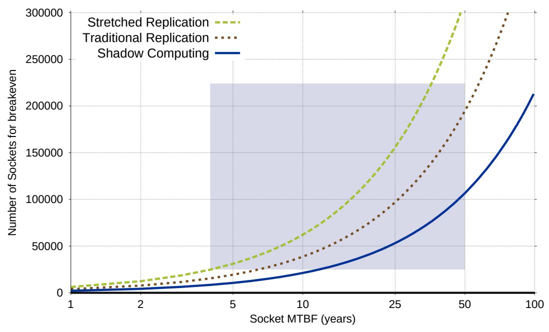
- I/O Bandwidth - This dictates how long it will take to write or recover a checkpoint.
- System Size - The number of total sockets.
- Socket MTBF - Reliability of a single socket in the computing system.
- Overhead Power - The overhead power consumed by the socket, as described in Section IV-B.

When comparing fault tolerant methods, we calculate the energy consumption and time using the power, failure and energy models described in the previous section.

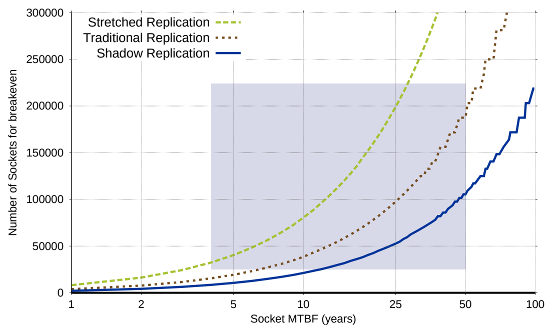
A. Scaling and Failure Rates

We compare fault tolerance efficiencies by identifying the breakeven point at which the replication technique is equivalent to that provided by coordinated checkpointing. We use two different breakeven metrics, the expected energy consumed and the expected time to solution. These are related to one another because energy is a function of time, but due to overhead power they are not equivalent. All of the area above the breakeven curve is where the replication technique is more efficient than coordinated checkpointing. Breakeven values are calculated by computing the energy and time required for coordinated checkpointing and comparing that to the energy and time required for the replication technique, when those values match that is the breakeven point.

Figure 3(a) shows the energy breakeven point varying system size and socket MTBF using a fixed checkpoint time of 15 minutes. These results show that shadow replication can provide a significant energy savings over traditional replication. For example when socket MTBF is 25 years traditional



(a) Breakeven Energy



(b) Breakeven Time

Fig. 3. Breakeven points for both energy and time given a fixed checkpoint time of 15 minutes and a system overhead power of 60%.

replication is viable at $96,700$ sockets whereas shadow replication is more efficient at $53,100$. This represents a 46% energy efficiency gain. Unfortunately, stretched replication turns out to be less energy efficient because of the increased time to solution and the presence of overhead power.

Shadow replication achieves this energy savings by slowing down the replicas, this raises the question of how this will effect the expected time to solution. Figure 3(b) plots the time to solution breakeven point, and shows that even though shadow replication slows the replicas the expected time to solution is actually shorter than that provided by traditional replication. For example, when socket MTBF is 25 years traditional replication is viable at $97,600$ sockets, whereas shadow replication is more efficient at just $52,700$ sockets, representing a 46% improvement in expected time to solution.

The improvement in time to solution is because shadow replication can utilize additional sockets while consuming the same power because the replicas are consuming less power. This is illustrated in Table I which shows the active socket counts allowable given a 20 mega-watt fixed power budget. Both stretched and shadow replication have the ability to use additional sockets because they reduce the power consumed by the individual sockets. Stretched replication reduces the power consumed by all processes equally whereas shadow replication only reduces the power consumed by the replica sockets. This is the reason that the expected time to completion of shadow replication outperforms traditional replication. Stretched replication is able to add additional nodes but because it also reduces the processor speed of the main processes, the time to solution is higher than both traditional and shadow replication.

In pure replication the total amount of work remains constant but the the number of sockets is half of that available to coordinated checkpointing. Our model assumes a strongly scaled application which is a fair comparison because each socket would have less work to accomplish in coordinated checkpointing, thus in a failure free case it would be faster than replication techniques. However, because with replication there are two sockets instead of one, the MTBF for the pair is greater than that provided in the single-socket case. The change in MTBF is what allows replication to out perform coordinated checkpointing at large scale. In shadow replication, instead of assuming half of the original sockets are replicas,

Overhead %	Method	# Sockets	# Main Sockets
60%	Checkpointing	$100,000$	$100,000$
60%	Traditional Replication	$100,000$	$50,000$
60%	Stretched $\alpha = 2.0$	$153,846$	$76,923$
60%	Shadow $\alpha = 2.0$	$124,998$	$62,499$
80%	Checkpointing	$100,000$	$100,000$
80%	Traditional Replication	$100,000$	$50,000$
80%	Stretched $\alpha = 2.0$	$120,230$	$60,115$
80%	Shadow $\alpha = 2.0$	$110,636$	$55,318$

TABLE I
AVAILABLE SOCKETS ASSUMING A 20 MEGA-WATT POWER LIMIT AND 200W PER SOCKET.

we calculate the energy optimal σ_b for $\alpha = 2.0$. Then "add" additional sockets remaining under the original power limit, but continuing to use half the sockets as replicas. Stretched replication is similar to shadow replication but both the replica and main use less power.

The conclusion is that shadow replication is both more energy efficient and produces solutions faster than traditional replication in power-limited systems. This is true for the majority of the exascale design space, illustrated by the region in the grey box in Figure 3. We assumed a fixed checkpoint time of 15 minutes [5] and a overhead power of 60% which are reasonable system parameters given expected exascale I/O bandwidth and increased system efficiencies. In the next sections we further relax these assumptions and study the models sensitivity to these parameters.

B. Scaling at Different Checkpoint I/O Rates

The checkpoint write times have a significant effect on the efficiency of coordinated checkpointing. These times are directly related to the available I/O bandwidth, as modeled in [13]. Figure 4 uses these models to determine the energy breakeven points for I/O bandwidth rates from 500GB/s to 50TB/s, representing a wide range of values for an exascale-class machine. For space reasons we only show results for shadow replication, other replication techniques follow a pattern similar to that in Figures 3(a) and 3(b). Shadow replication is viable for all but very extreme levels of I/O bandwidth.

C. Scaling at Different Overhead Power

Table I illustrated that the number of available sockets decreases as the percentage of overhead power increases. Shadow

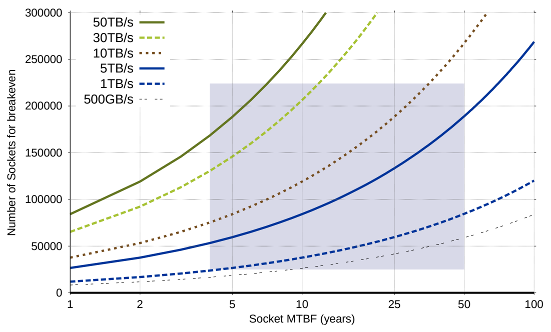


Fig. 4. Shadow replication energy breakeven for different I/O bandwidths. Assumes 16Gb per socket.

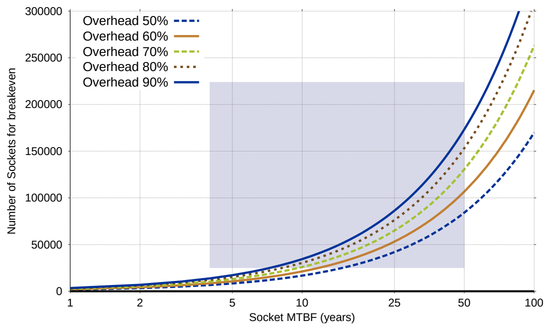


Fig. 5. Shadow replication energy breakeven for various overhead power percentages. Checkpoint time of 15 minutes.

replication can only reduce dynamic power consumption, leaving it with less power headroom to improve efficiency. This means it can take advantage of fewer main sockets as the available power headroom decreases. Figure 5 shows the affect overhead power has upon the energy breakeven point. As the power overhead increases the potential energy savings also decreases, moving the breakeven point further out into the exascale domain. The conclusion is that overhead power does have an effect upon shadow replication but even if the overhead is 100% it will be no worse than traditional replication. It is expected that future hardware will reduce this overhead making shadow replication more efficient.

D. Energy Savings

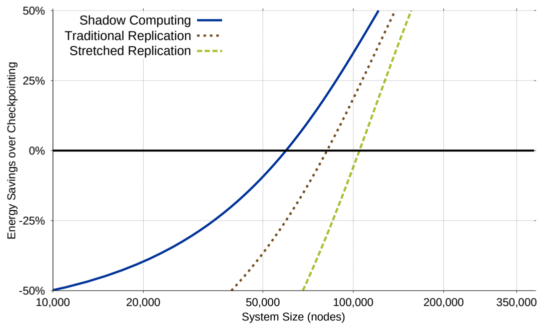


Fig. 6. Energy savings achieved by all replication method for a system with 1TB/s I/O bandwidth and 25 Year MTBF.

Figure 6 shows the relative energy savings for a system with 1TB/s I/O bandwidth and 25 Year MTBF. The trend is consistent with that observed in the breakeven graphs and shows that shadow replication consistently consumes about 20% less energy than that consumed by pure replication. For space reasons we have only shown one system configuration but this trend is consistent across different configurations, although as we have previously shown, the breakeven location changes. In this graph the breakeven point is the x-axis.

VI. Implementation and Evaluation

To provide additional insight into the energy consumption of the methods described in this paper and validate the results of our models, we implemented the replication techniques in MPI and measured the energy usage of several applications. Replication has previously been implemented within MPI [15] but existing implementations did not have the ability to change the execution speeds of the replicas. We implemented replication as a MPI profiling library for OpenMPI that allows us to execute unmodified MPI applications using the replication techniques discussed in this paper. To determine optimal execution speeds for shadow computing, an estimate of the total amount of work is needed. Most production job queue systems require applications to estimate their execution time before submitting a job and we use that estimate to determine the execution speeds.

To measure the power consumption of each of these techniques we use a computing cluster that is equipped with component-level power measurement instrumentation. This cluster contains 104 nodes, each with a AMD Llano Fusion, which is a 4-core AMD K10 x86 paired with a 400-core Radeon HD 6550D. Additional details on the power measurement and validation of the system can be found in [11].

To evaluate the different methods of fault tolerance we selected three different MPI applications. The first one is a production application called LAMMPS [14] which is a molecular dynamics code. The two others are mini-applications from Sandia's mantevo suite [16]: HPCCG (a conjugate gradient solver) and miniFE (an implicit finite element method). These applications are important as they represent a range of computational techniques, are frequently run at very large scales on leadership class systems, and represent key simulation workloads for the U. S. Department of Energy.

A. Experimental Results

The purpose of these small-scale experiments are to demonstrate that the power-aware replication techniques can provide measurable energy savings for actual HPC applications. In Figure 7 the average total energy consumption of multiple application runs are shown for each replication technique, normalized to the energy consumed by traditional replication. This shows that power-aware replication techniques reduce overall energy but also demonstrates that the amount of savings is application dependent, as previous studies have found [10]. HPCCG and miniFe show the maximum energy savings. This is because they are simple applications that are processor bound. Looking at LAMMPS, which is a production application, one can see that the energy savings follows the

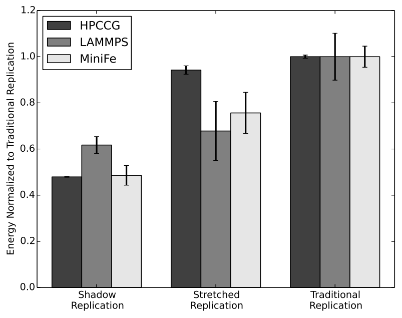


Fig. 7. Experimental results of the energy savings achieved by different replication schemas.

same trend but the amount of energy saved is less than for the mini-applications. While it is hard to predict exactly what the energy savings will be, it is clear that our proposed techniques have the potential to save energy.

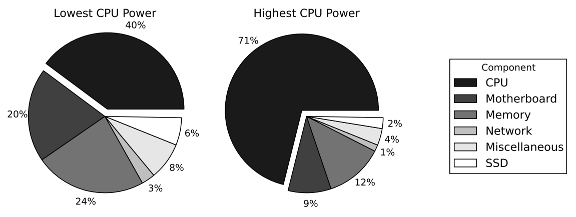


Fig. 8. Component level energy usage for LAMMPS

To confirm our assumptions about overhead power we looked at the component level energy usage over runs of real applications. In Figure 8 we show the percentage of energy consumption by component for multiple runs of the LAMMPS application. The first chart is the energy consumed when running LAMMPS at the lowest possible execution speed. In this case, the CPU consumes 40% of the overall energy. The second chart shows the energy consumption when running at full power in which the CPU consumes 71% of the overall energy. From this the estimated amount of overhead power is 67%, we observed a similar pattern for other applications, concluding that overhead power in our system is 60-67%.

VII. Conclusions and Future Work

In this work we show under what circumstances replication is the most energy and time efficient fault tolerance mechanism available. Furthermore, we show the benefit of power-aware modifications to replication. Replication can be made 40% more time and energy efficient using a simple protocol termed *shadow replication*. This savings makes replication a viable fault tolerance solution through the majority of the exascale-class design space. Additionally, we demonstrate at small scale that the proposed modifications actually produce energy savings for actual HPC workloads. Most importantly, this work demonstrates the need to consider power as a first order design constraint in fault tolerance methods for exascale systems.

While these results are promising, there are several avenues of future work being pursued. First, due to the reduced speed of the replicas one concern is the growth of the message queues between replica and their leaders, which occupy memory on the replica nodes. The rate of this queue growth depends highly upon an application message rate. While we are still investigating possible solutions, the most promising draws an analogy between these queues and the message logs used in uncoordinated checkpointing techniques. The idea, therefore, is to use an applications send-determinism property [7] to reduce the size of message logs while avoiding cascading rollbacks. Lastly, we are continuing to explore other power-aware modifications to replication and ways to efficiently pair replication with checkpointing.

References

[1] S. Ahern, A. Shoshani, K.-L. Ma, A. Choudhary, T. Critchlow, S. Klasky, V. Pascucci, J. Ahrens, E. W. Bethel, H. Childs, J. Huang, K. Joy, Q. Koziol, G. Lofstead, J. S. Meredith, K. Moreland, G. Ostrouchov, M. Papka, V. Vishwanath, M. Wolf, N. Wright, and K. Wu. Scientific discovery at the exascale, a report from the doe ascr 2011 workshop on exascale data management, analysis, and visualization, 2011.

[2] J. Ahn. 2-step algorithm for enhancing effectiveness of sender-based message logging. In *SpringSim '07: Proceedings of the 2007 spring simulation multiconference*, pages 429–434, 2007.

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

[4] 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*, 2012.

[5] K. Ferreira, J. Stearley, J. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. 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, New York, NY, USA, 2011. ACM.

[6] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5), 2010.

[7] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello. Uncoordinated checkpointing without domino effect for send-deterministic mpi applications. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, Washington, DC, USA, 2011. IEEE Computer Society.

[8] Q. Jiang and D. Manivannan. An optimistic checkpointing and selective approach for consistent global checkpoint collection in distributed systems. In *Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium*, Mar. 2007.

[9] D. B. Johnson and W. Zwaenepoel. Recovery in distributed systems using asynchronous and checkpointing. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, 1988.

[10] J. Laros, K. Pedretti, S. Kelly, W. Shu, and C. Vaughan. Energy based performance tuning for large scale high performance computing systems. In *20th High Performance Computing Symposium*, Orlando Florida, 2012.

[11] J. Laros, P. Pokorny, and D. DeBonis. Powerinsight - a commodity power measurement capability. In *The Third International Workshop on Power Measurement and Profiling in conjunction with IEEE IGCC 2013*, Arlington Va, 2013.

[12] E. Meneses, O. Sarood, and L. V. Kalé. Assessing energy efficiency of fault tolerance protocols for hpc systems. In *SBAC-PAD*, 2012.

[13] 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.*, 2007.

[14] S. J. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal Computation Physics*, 117, 1995.

[15] R. Riesen, K. Ferreira, J. Stearley, R. Oldfield, J. Laros, K. Pedretti, and R. Brightwell. Redundant computing for exascale systems. Technical Report SAND2010-8709, Sandia National Laboratories, 2010.

[16] Sandia National Laboratory. Mantevo project home page. https://software.sandia.gov/mantevo, 2010.