Shadow Computing

An Energy-Aware Resiliency Scheme for Cloud 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 cloud 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 an energy-optimal solution that can be applied within the cloud computing environment to provide fault tolerance while minimizing energy consumption.

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 volumes of data [11]. Addressing these challenges requires radical changes in the way computing is delivered to support the computing requirements and characteristics of these applications [22]. 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 of these systems

A direct implication of these emerging trends is the need to address the difficult challenge of minimizing power consumption. Furthermore, the fault rates are expected to dramatically increase, possibly by several orders of magnitude [23, 25]. Figure 2 shows the system mean time between failures and the number of faults as a function of the number of nodes in the system [21]. 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 storage components, but will also grow out of the necessity to use advanced technology, at lower voltage levels, and deal with undesirable aging effects as they become significant [23]. Addressing this concern brings about unprecedented resiliency challenges, which puts in question the ability of next generation cloud computing infrastructure to continue operation in the presence of faults without compromising the requirements of the supported applications.

The current response to faults in existing systems consists in 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, however, 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. Note that, in some situations, several components of the software environment associated with the failed application may have to be restarted.

Given the anticipated failure rate in high-performance, large-scale compute- and data-intensive environments, 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. The results of a study carried out at Sandia National Laboratories to evaluate the overhead incurred by checkpointing, depicted in Figure 1, clearly show that beyond 50,000 nodes the application spends only a fraction of the elapsed time performing useful computation.

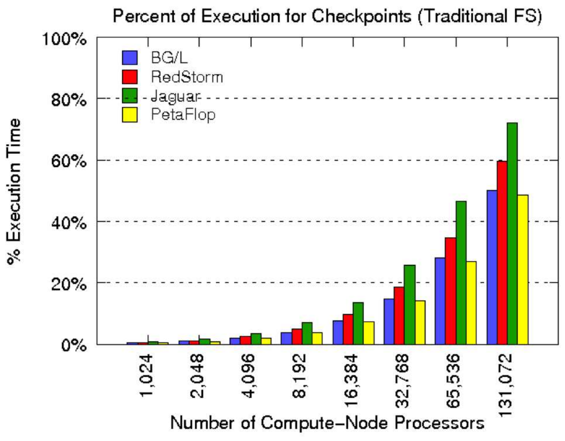


Figure 1: Percent of time used to perform checkpoints as the number of nodes increase.

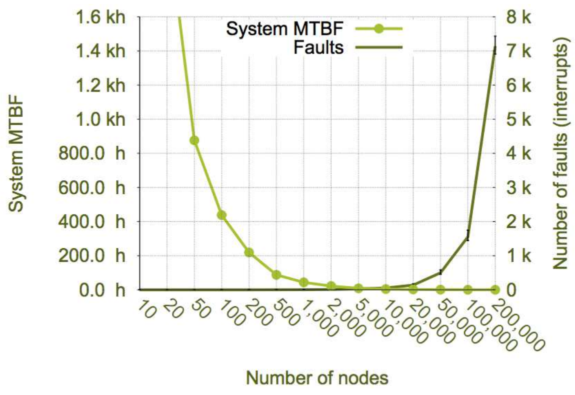


Figure 2: Effect on system MTBF as number of nodes increase.

Current fault-tolerant frameworks are typically designed to handle single errors, whereas computation in large-scale, high-performance computing environments is likely to face multiple different types of errors, often concurrently. Even in the case of single errors, current fault-tolerant approaches apply the same technique, mainly checkpoint-and-restart over the entire duration of the execution, to handle all types of faults, including permanent node crashes, transient computing errors, and input-out 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 achieve scalable resiliency in future large-scale, high-performance cloud computing infrastructure. To this end, we propose a new energy-aware "shadow computing" scheme, as an efficient and scalable alternative to checkpointing and rollback recovery based techniques.

The basic idea the shadow computing computation 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 process is a replicate of the main process. To overcome failure while minimizing energy consumption, the shadow runs concurrently with the main process, but at different computing node and at a reduced processor speed. The successful completion of the main process results in the immediate termination of the shadow process. In case the main process fails, however, two actions are simultaneously undertaken by the system. First, the shadow process immediately takes over the role of the main process and resumes computation at increased speed, without disturbing other related processes. Second, a new shadow process is initiated in anticipation of future failures. It is worth noting, that failure of the shadow process does not impact the behavior of the main process and simply results in the activation of a new shadow process to replace the failing one.

In order to fully harness the potential of shadow computing to efficiently deal with failures, an optimization model is proposed to derive the execution speed of the main process and the prior- and post-failure execution speeds of the shadow process. The derived execution speeds 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 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 have adverse impact on the resiliency 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 reminder of the paper is organized as follows: Section II work related to fault-tolerance in large-scale high-performance computing systems. Section III presents the energy optimization model used to derive the different execution speeds of the main process and its shadow. Section IV presents the energy optimal shadow computing model. Section V discusses the results of a comparative analysis of the optimal energy-aware shadow computing scheme to other fault-tolerant schemes. Section VI 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, 12, 15, 16, 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, 14, 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 [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 [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.

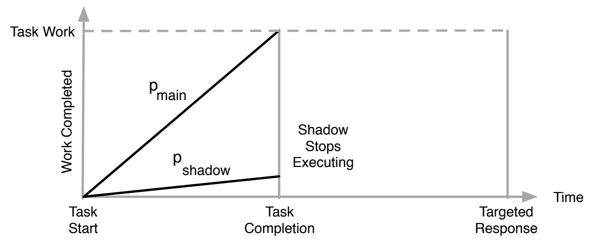
In addition to its significant energy saving, shadow computing has potential to significantly increase an application's mean time to interrupt (MTTI). This model can be further enhanced to allow for diversity through the execution of multiple shadow processes in different geographical environments, using 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

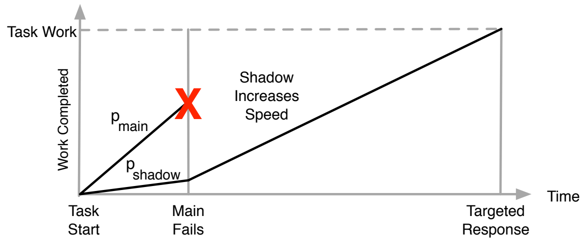
In this section, we provide an overview of the shadow computing execution model, under different failure scenarios. We also discuss the mapping of the processes to the computing infrastructure to ensure fault-tolerance to failure. Finally, we present the basic data structure that enables efficient communication 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.



(a) Shadow Computing Case of no failure



(b) Shadow Computing Case of failure

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

[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

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 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 clound computing, fault-isolation uses the virtual computing 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

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

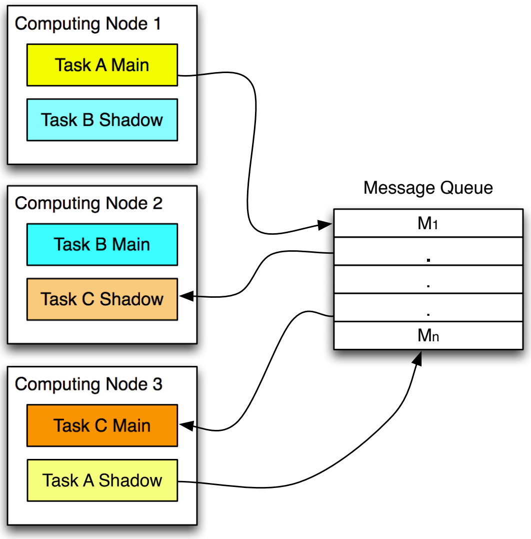


Figure 3: Example Process Mapping

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.

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

[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.

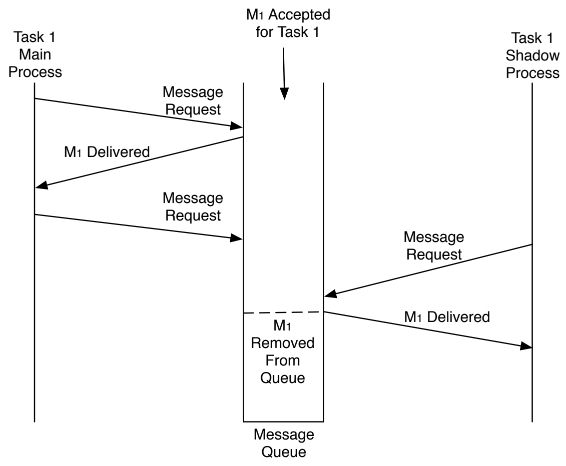


Figure 4: Example Message Delivery

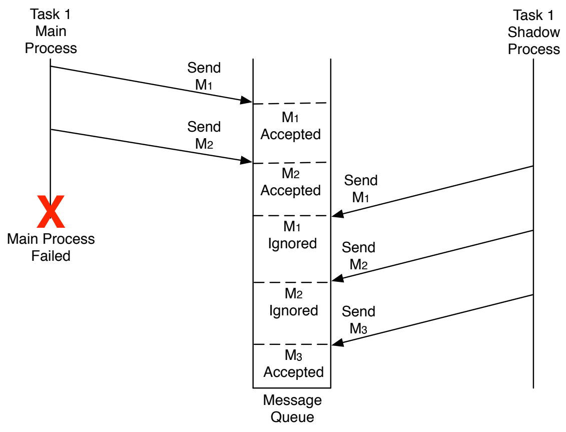


Figure 5: Example Message Receiving

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.

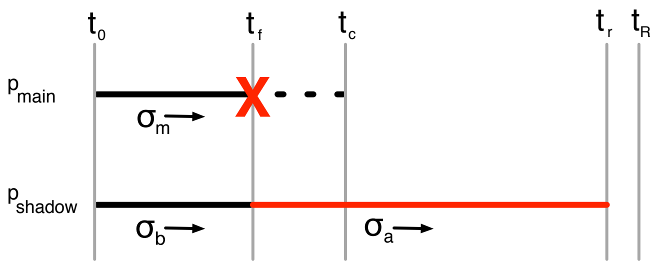


Figure 6: Overview of Shadow Computing

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 **??** 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 CLOUD COMPUTING

Cloud computing providers seek to find a balance between the execution speed and fault resiliency of a given configuration in order to maximize their profit. "Shadow computing" provides a mechanism to make this tradeoff by optimizing the energy consumption of fault tolerance given a task's targeted 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.

The remainder of this section presents a solution to finding σ_m, σ_b and σ_a for energy optimal replication. We start by specifying our failure and power model then show how we use numerical analysis techniques to find optimal values.

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 two variables, σ_m and σ_b.

In addition to minimizing the objective energy function we also must define several constraint functions for our minimization problem. These constraints ensure that our model obeys the system limitations such as maximum speed but also ensure that the work is completed by the targeted response time. The first set of constraints simply bound the values of σ_b and σ_m, $0 \leq \sigma_b \leq \sigma_{max}$ and $0 \leq \sigma_m \leq \sigma_{max}$

Next we bound σ_m such that the main process will finish at or before the targeted response time R,

$$\sigma_m W/R \tag{8}$$

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.

$$t_c * \sigma_b + (R - t_c) * \sigma_{max} \geq W \tag{9}$$

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_m and σ_b that minimizes the following function.

$$\begin{aligned}
&\int_{t=0}^{t_c} (\sigma_m^2 t + \sigma_b^2 t) e^{-\lambda t} dt \\
&+ \int_{t=0}^{t_c} [(W - \sigma_b * t_f)/(R - t_f)]^2 (R - t) e^{-\lambda t} dt \\
&+ (1 - \int_{t=0}^{t_c} e^{-\lambda t} dt)(\sigma_m^2 t_c + \sigma_b^2 t_c)
\end{aligned} \tag{10}$$

Also note that $t_c = W\sigma_m$ because the amount of work, W, is given and main process execution speed is defined as σ_m. After solving equation 10 we use numerical analysis techniques for finding the optimal execution speeds, σ_m and σ_b.

Our optimization outputs the main processes execution speed, σ_m, and 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.

6. ANALYSIS

Using the energy model and constraints described in the previous sections we compare the energy consumed by the optimal energy shadow, pure replication, stretched replication and delayed re-execution. For the optimal energy shadow we rely upon numerical non-linear optimization techniques to find the speeds, σ_m and σ_b, that minimize the consumed energy. All results presented used the Minimize function available in Mathmatica.

6.1 Replication vs. Re-execution vs. Optimal Energy Shadow

Despite improvements in the checkpointing and rollback scheme it has been shown to not scale as the probability of failure increases. Given this it is has been suggested that pure process replication or process re-execution should be used to provide fault tolerance. In this section we will compare the energy consumed by pure replication, stretched replication, re-execution and optimal energy shadow.

Pure replication is represented in our model by simply letting all the speeds equal the maximum speed, $\sigma_m = \sigma_b = \sigma_a = \sigma_{max}$. For stretched replication we let all execution speeds be equal $\sigma_m = \sigma_b = \sigma_a = W/R$. Process re-execution simply re-executes a process if the main process fails. Using our energy model this is equivalent to setting σ_b to zero, therefore no work is done by the shadow until failure. There are several different ways of selecting the execution speed of the main process but for our analysis we let it be the slowest possible speed that allows for re-execution given σ_{max}. Therefore, for re-execution $\sigma_m = W/(R - (W\sigma_{max}))$.

We find that optimal energy shadow computing consistently out performs or matches the energy consumed by all other schemes while continuing to deliver the same fault tolerance level. Figure 9 demonstrates this by showing the energy consumption as a function of the rate of failure, λ. In this analysis we vary the targeted response, R, specifically we vary $R = \alpha * W/\sigma_{max}$, where $\alpha 1$. Without loss of generality we let $\sigma_{max} = 1$.

The first observation is that when $\alpha = 1.00$ all fault tolerant schemes consume the same energy. This is expected since all processes must work at σ_{max} to complete by the targeted response time, R. However, as we increase α, thus increasing the slack, we observe that optimal energy shadow saves as much as 58% of the energy consumed by pure replication. The other important observation is that stretched replication uses almost the same amount of energy as that of the optimal energy shadow. For all values of α optimal energy shadow and stretched replication converge as the rate of failure increases. The reason for this convergence is that as the likelihood of the main process failing increases the optimal shadowâĂŹs execution speeds evenly spreads the work out, thus approaching the same speed used in replication, W/R.

Re-execution can only be achieved when the targeted response time allows enough time to re-execute the task. In other words the system must have enough slack to re-execute the task. Because we have fixed $\sigma_{max} = 1$, we can only achieve re-execution when $\alpha \geq 2$. It can observed in the last figure that both stretched replication and optimal energy shadow consistently outperforms re-execution.

6.2 System Slack

Assuming a fixed amount of work, W, there are two ways of producing slack in the system, one is to increase the targeted response time and the other is to increase σ_{max}. The more slack in the system the more energy optimal shadow computing can save however when compared to replication and re-execution the way slack is generated has different effects.

Neither pure replication nor stretched replication can benefit from increasing the maximum speed, σ_{max}, given a targeted response time R. This is because replication will always pick a consistent speed σ_{max} or $\sigma = W/R$. Increasing σ_{max} has no impact on the speed of execution in replication thus it will have no impact on the energy consumed. However, increasing σ_{max} reduces the energy consumption in optimal energy shadow computing. This is directly related to the constraint ensuring that in the worst case the shadow will execute at the maximum speed possible after failure to achieve the targeted response time. By increasing σ_{max} it allows the shadow to execute at a slower speed before failure because of the ability to execute faster after failure. This has the effect of allowing shadow computing to save energy as the maximum available speed increases. This can be observed in Figure 5, each graph represents the effect of increasing σ_{max}.

Similar to shadow computing, re-execution also benefits from having faster execution speeds because it can choose a slower execution speed for its main process. In all cases optimal energy shadow computing outperforms replication and re-execution. However, observe that as the rate of failure increases optimal converges to stretched replication but as the rate of failure decreases it converges to re-execution.

If the maximum execution speed is increasing but the amount of work and targeted response time is the same then one should choose optimal energy shadow. This is because it is the only scheme described that can take full advantage of slack introduce by increasing the execution speed. Thus you can save energy by harnessing the fact that your underling architecture can go faster.

6.3 Job Size

For this analysis we wanted to understand the relationship between failure rates and job size as it relates to energy consumption. To do this we rewrite lambda as the mean time between failure (MTFB). Because we are using the exponential probability distribution we know the mean value is given by $1/\lambda$. We then let this represent the number of seconds between failure such that the MTFB is the number of days between failure. Similarly, we scale the size of the job to be number hours/days. This allows us to observe the energy savings in a realistic context.

In Figure 5, we show the energy savings using energy optimal shadow computing over the energy consumed using pure replication. The first observation is that we consistently save 20-50% energy. We show these results for various values of Îś; again it can be observed that, as the slack increases, shadow computing achieves more energy saving. Conversely, as the size of the job increases, the savings decrease; this is due to a decrease of the slack in the system.

7. CONCLUSION

In this paper we have introduced shadow computing, an energy efficient method to provide fault tolerate execution without the limitations of checkpointing. We then compared this to other known methods, replication and re-execution, and concluded that shadow computing is always more energy efficient. We also observed that the amount of energy saving is highly dependent upon the rate of failure and the amount of slack present in the system.

Fully harnessing the potential of shadow computing to deal with failures brings about several challenging questions that need to be addressed: How can this concept be used to improve fault detection and layer coordination, understanding faults and silent errors and improving situational awareness? What level of synchronization is required between the main process and its associated shadow processes to minimize impact on other application processes? What state, if any, must be saved to ensure âĂIJsmoothâĂİ transition to the primary shadow process upon failure of the main process? Future work will be focused on investigating these questions for different types of failure to better understand the advantages and limitations of this approach to achieve high levels of fault-tolerance in extreme scale cloud computing environments.

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.

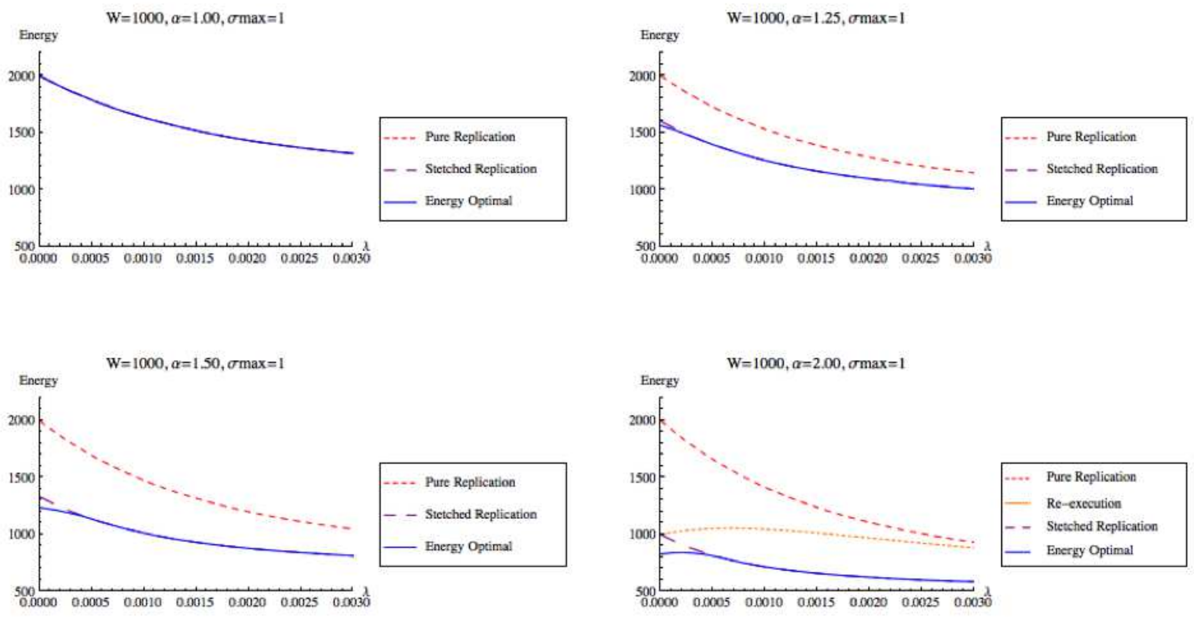


Figure 7: Energy comparison between optimal energy, pure replication and re-execution.

[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] J. F. Gantz, D. Reinsel, C. Chute, W. Schlichting, J. Mcarthur, S. Minton, I. Xheneti, A. Toncheva, and A. Manfrediz. IDC - The Expanding Digital Universe: A Forecast of Worldwide Information Growth Through 2010. Technical report, Mar. 2007.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[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.

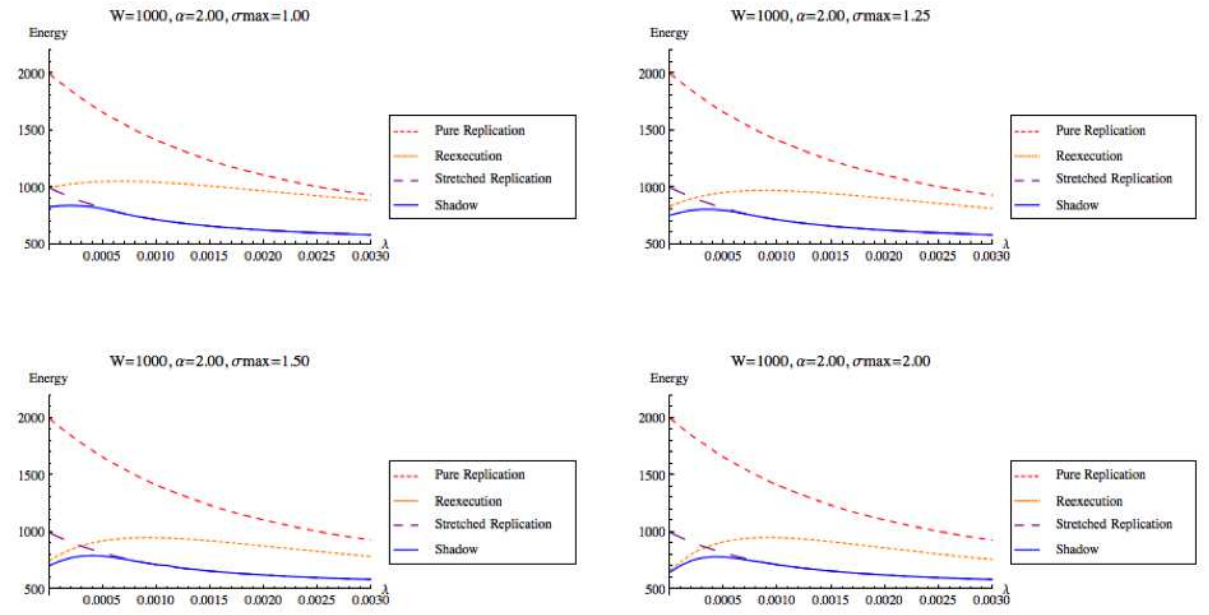


Figure 8: Energy comparison between optimal energy, pure replication and re-execution, vary alpha.

[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.

[25] J. Torrellas. Architectures for extreme-scale computing. *Computer*, 42(11):28–35, Nov. 2009.

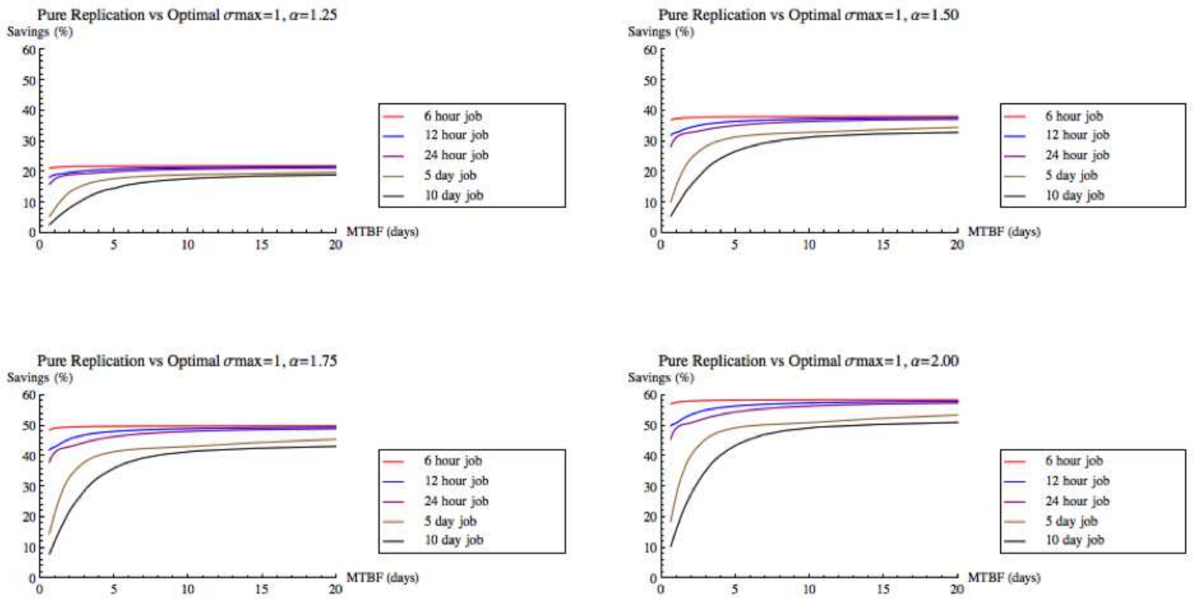


Figure 9: Energy comparison between optimal energy, pure replication and re-execution, vary alpha.