ADAPTIVE AND POWER-AWARE FAULT TOLERANCE FOR FUTURE EXTREME-SCALE COMPUTING

by

Xiaolong Cui

Bachelor of Engineering

Xi'an Jiaotong University

2012

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of

Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2016

UNIVERSITY OF PITTSBURGH

COMPUTER SCIENCE DEPARTMENT

This proposal was presented

by

Xiaolong Cui

Dr. Taieb Znati, Department of Computer Science, with joint appointment in Telecommunication Program, University of Pittsburgh

Dr. Rami Melhem, Department of Computer Science, University of Pittsburgh

Dr. John Lange, Department of Computer Science, University of Pittsburgh

Dr. Esteban Meneses, School of Computing, Costa Rica Institute of Technology

Dissertation Advisors: Dr. Taieb Znati, Department of Computer Science, with joint appointment in Telecommunication Program, University of Pittsburgh,

Dr. Rami Melhem, Department of Computer Science, University of Pittsburgh

ADAPTIVE AND POWER-AWARE FAULT TOLERANCE FOR FUTURE EXTREME-SCALE COMPUTING

Xiaolong Cui, PhD

University of Pittsburgh, 2016

As the demand for computing power continue to increase, both HPC community and Cloud service provides are building larger computing platforms to take advantage of the power and economies of scale. On the HPC side, a race is underway to build the world's first exascale supercomputer to accelerate scientific discoveries, big data analytics, etc. On the Cloud side, major IT companies are all expanding large-scale datecenters, for both private usage and public services. However, aside from the benefits, several daunting challenges will appear when it comes to extreme-scale.

This thesis aims at simultaneously solving two major challenges, i.e., power consumption and fault tolerance, for future extreme-scale computing systems. We come up with a novel power-aware computational model, referred to as Lazy Shadowing, to achieve high-levels of resilience in extreme-scale and failure-prone computing environments. Different approaches have been studied to apply this model. Accordingly, precise analytical models and optimization framework have been developed to quantify and optimize the performance, respectively.

In this work, I propose to continue the research in two aspects. Firstly, I propose to develop a MPI-based prototype to validate Lazy Shadowing in real environment. Using the prototype, I will run benchmarks and real applications to measure its performance and compare to state-of-the-art approaches. Then, I propose to further explore the potential of Lazy Shadowing and improve its efficiency. Based on the specific system configuration, application characteristics, and QoS requirement, I will study the impact of process mapping the viability of partial shadowing.

TABLE OF CONTENTS

1.0 INTRODUCTION

As our reliance on IT continues to increase, the complexity and urgency of the problems our society will face in the future drives us to build more powerful and accessible computer systems. Among the different types of computer systems, High Performance Computing (HPC) and Cloud Computing systems are the two most powerful ones. For both of them, the computing power attributes to the massive amount of parallelism, which is supported by the massive amount of computing cores, memory modules, communication devices, storage components, etc.

Since CPU frequency flattens out in early 2000s, parallelism has become the "golden rule" to boost performance. In HPC, Terascale performance was achieved in the late 90s with fewer than 10,000 heavyweight single-core processors. A decade later, petascale performance required about ten times processors with multiple cores on each processor. Nowadays, a race is underway to build the world's first exascale machine to accelerate scientific discoveries and breakthroughs. It is projected that an exascale machine will achieve billion-way parallelism by using one million sockets each supporting 1,000 cores.

Similar trend is happening in Cloud Computing. As the demand for Cloud Computing accelerates, cloud service providers will be faced with the need to expand their underlying infrastructure to ensure the expected levels of performance, reliability and cost-effectiveness. As a result, lots of large-scale data centers have been and are being built by IT companies to exploit the power and econies of scale. For example, Microsoft, Google, Facebook, and Rackspace have hundreds of thousands of web servers in dedicated data centers to support their business.

Unfortunately, several challenging issues come with the increase in system scale. As today's HPC and Cloud Computing systems grow to meet tomorrow's compute power demand,

the behavior of the systems will be increasingly difficult to specify, predict and manage. This upward trend, in terms of scale and complexity, has a direct negative effect on the overall system reliability. Even with the expected improvement in the reliability of future computing technology, the rate of system level failures will dramatically increase with the number of components, possibly by several orders of magnitude. At the same time, the rapid growing power consumption, as a result of the increase in system components, is another major concern. At future extreme-scale, failure would become a norm rather than an exception, driving the system to significantly lower efficiency with unprecedented amount of power consumption.

1.1 PROBLEM STATEMENT

The system scale needed to address our future computing needs will come at the cost of increasing complexity, unpredictability, and operating expenses. As we approach future extreme-scale, two of the biggest challenges will be system resilience and power consumption, both being direct consequences of the increase in the number of components.

Regardless of the reliability of individual component, the system level reliability will continue to decrease as the number of components increases. It is projected that the Mean Time Between Failures (MTBF) of future extreme-scale systems will be at the order of hours or even minutes, meaning that many failures will occur every day. Without an efficient fault tolerance mechenism, faults will be so frequent that the applications running on the systems will be continuously interrupted, requiring the execution to be restarted every time there is a failure.

Also thanks to the continuous growth in system components, there has been a steady rise in power consumption in large-scale distributed systems. In 2005, the peak power consumption of a single supercomputer reached 3.2 Megawatts. This number was doubled only after 5 years, and reached 17.8 Megawatts with a machine of 3,120,000 cores in 2013. Recognizing this rapid upward trend, the U.S. Department of Energy has set 20 megawatts as the power limit for future exascale systems, challenging the research community to provide a 1000x

improvement in performance with only a 10x increase in power. This huge imbalance makes system power a leading design constraint on the path to exascale.

Today, two approaches exist for fault tolerance. The first approach is rollback recovery, which rolls back and restarts the execution every time there is a failure. This approach is often equiped with checkpointing to periodically save the execution state to a stable storage so that execution can be restarted from a recent checkpoint in the case of a failure. Although checkpointing is the most widely used technique in today's HPC systems, it is strongly believed that it may not scale to future extreme-scale systems. Given the anticipated increase in system level failure rates and the time to checkpoint large-scale compute-intensive 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 MTBF. Consequently, applications will make little forward progress, thereby reducing considerably the overall system efficiency.

The second approach, referred to as process replication, exploits hardware redundancy and executes multiple instances of the same task in parallel to overcome failure and guarantee that at least one task instance reaches completion. Although this approach is extensively used to deal with failures in Cloud Computing and mission critical systems, it has never been used in any HPC system due to its low system efficiency. To replicate each process, process replication requires at least double the amount of compute nodes, which also increases the power consumption proportionally.

Previous studies show that neither of the two approaches may be efficient for future extreme-scale systems. And unfortunately, neither of them addresses the power cap issue. Achieving high resilience to failures under strict power constraints is a daunting and critical challenge that requires new computational models with scalability, adaptability, and power-awareness in mind.

1.2 RESEARCH OVERVIEW

There is a delicate interplay between fault tolerance and power consumption. Checkpointing and process replication require additional power to achieve fault tolerance. Conversely, it has been shown that lowering supply voltages, a commonly used technique to conserve power, increases the probability of transient faults. The trade-off between fault free operation and optimal power consumption has been explored in the literature. Limited insights have emerged, however, with respect to how adherence to application's desired QoS requirements affects and is affected by the fault tolerance and power consumption dichotomy. In addition, abrupt and unpredictable changes in system behavior may lead to unexpected fluctuations in performance, which can be detrimental to applications QoS requirements. The inherent instability of extreme-scale computing systems, in terms of the envisioned high-rate and diversity of faults, together with the demanding power constraints under which these systems will be designed to operate, calls for a reconsideration of the fault tolerance problem.

In this thesis, our research objective is to simultaneously address the power and resilience challenges for future extreme-scale systems so that both system efficiency and application QoS are guaranteed. To this end, we propose an adaptive and power-aware computational model, referred to as Lazy Shadowing, as an efficient and scalable alternative to achieve high-levels of resilience, through forward progress, in extreme-scale, failure-prone computing environments.

Previously, we have formally defined the computational model, studied possible techniques to realize and optimize the idea, and built analytical models for performance evalution. Next, I propose to continue the study of Lazy Shadowing in two aspects. Firstly, I propose to implement a prototype of Lazy Shadoing in the context of Message Passing Interface (MPI), to validate our computational model as well as measure its performance in real environment. Secondly, I propose to study the possibility of "smart shadowing" which further reduces the cost of shadowing by considering the specific system configuration, application characteristics, and QoS requirement.

1.2.1 Lazy Shadowing: a novel fault-tolerant computational model (completed)

The basic tenet of Lazy Shadowing is to associate with each main process a suite of shadows whose size depends on the "criticality" of the application and its performance requirements. Each shadow process is an exact replica of the original main process. To tolerate failures, the main process and its associated shadow processes will execute in parallel, but on different compute nodes. The shadows initially execute at a reduced rate via Dynamic Voltage Frequency and Scaling (DVFS) to save power. If the main process completes the task succesfully, we will terminate the shadows immediately. If the main process fails, however, we will promote one of the shadow processes to be a new main process and possibly increase its execution rate to mitigate delay. This continues until the task completes.

Given that the failure rate of an individual node is much lower than the aggregate system failure, it is very likely that the main process will always complete its execution successfully, thereby achieving fault tolerance at a significantly reduced cost of power. Consequently, the high probability that shadows never have to complete the full task, coupled with the fact that they initially only consume a minimal amount of power, dramatically increases a power-constrained systems tolerance to failure.

The major challenge in Lazy Shadowing resides in determining jointly the execution rates of all task instances, both before and after a failure occurs, with the objective to optimize performance, resilience, and/or power consumption. As a case study, we develop a reward-based analytical framework to derive the optimal execution rates for minimizing energy consumption under strict completion time constraints.

1.2.2 Applying Lazy Shadowing to extreme-scale systems (completed)

Enabling Lazy Shadowing for resiliency in extreme-scale computing brings about a number of challenges and design decisions that need to be addressed, including the applicability of this concept to a large number of tasks executing in parallel, the effective way to control shadows execution rates, and the runtime mechanisms and communications support to ensure efficient coordination between a main and its shadow. Taking into consideration the main characteristics of compute-intensive and highly-scalable applications, we design two novel

techniques, referred to as shadow collocation and shadow leaping, in order to achieve high tolerance to failures while minimizing delay and power consumption.

To control the processes' execution rate, DVFS can be applied while each process resides on one core exclusively. The effectiveness of DVFS, however, may be markedly limited by the granularity of voltage control, the number of frequencies available, and the negative effects on reliability. An alternative is to collocate multiple processes on each core while keeping all the cores executing at maximum frequency. Then time sharing can be used to achieve the desired execution rates for each collocated process. Since this approach collocates multiple processes on a core, it simultaneously reduces the number of compute nodes and the power consumption.

Furthermore, we identify a unique opportunity that allows the lagging shadows to benefit from the faster execution of the mains, without incurring extra overhead. Specifically, when a failure occurs, Lazy Shadowing takes advantage of the recovery time and leaps forward the shadows by copying states from the mains. This technique not only achieves forward progress for the shadows at minimized power and delay, but also reduces the recovery time after each failure.

1.2.3 lsMPI: implementation of Lazy Shadowing in MPI (partially completed)

Though Lazy Shadowing has been shown to scale to future extreme-scale systems with our analytical models, a real implementation is still necessary for validation and performance measurement in real systems. We plan to implement a prototype of Lazy Shadowing as a runtime for Message Passing Interface (MPI), which is the de facto programming paradigm for HPC. Instead of a full-feature MPI implementation, the runtime is designed to be a separate layer between MPI and user application, in order to take advantage of existing MPI performance optimizations that numerous researches have spent years on. The runtime will spawn the shadow processes at initialization phase, manage the coordination between main and shadow processes during execution, and guarantees consistency for messages and non-deterministic events. With this implementation, we will perform thorough experiments measuring its runtime overhead as well as performance under failures.

1.2.4 Smart shadowing (future)

Lazy Shadowing is a flexible and adaptive computational model that deserves further investigation. Previous studies have shown that different nodes tend to have different failure probabilities, e.g., 20% of the nodes account for 80% of the failures. The reason is complicated and may attribute to the manufacture process, heterogenous architecture, environment factors (e.g. temperature), and/or workloads. I propose to apply machine learning techniques to learn the heterogenity in failure distributions among a given system's nodes. Then I will study how the mapping from processes to physical cores can impact the performance and cost dichotomy. In addition, I will further consider allocating different number of shadow processes for different tasks to reduce cost while maintaining performance.

1.3 CONTRIBUTIONS

This thesis makes the following contributions:

- Development of an adaptive and power-aware computational model for efficient and scalable fault tolerance in future extreme-scale computing systems.
- Comprehensive and accurate analytical models to quantify the expected completion time and energy consumption in both HPC and Cloud Computing systems.
- A fully functional implementation of Lazy Shadowing for Message Passing Interface
- Exploration of Lazy Shadowing's potential to adapt to different environments, workloads, and QoS requirements.

1.4 OUTLINE

The rest of this proposal is organized as follow: Chapter 2 introduces fault tolerance and power management in large-scale distributed systems. In Chapter 3, we formally define Lazy Shadowing computational model and build analytical model for performance evaluation.

In Chapter 4, we explore techniques to efficiently apply Lazy Shadowing in extreme-scale systems. Implementation issues are discussed in Chapter 5. Adaptivity and smart shadowing are discussed in Chapter 6. Chapter 7 and 8 lists the timeline and concludes the proposal, respectively.

2.0 BACKGROUND

Rollback recovery is the dominant mechanism to achieve fault tolerance in current HPC environments [Elnozahy and et. al., 2002]. In the most general form, rollback recovery involves the periodic saving of the execution state (checkpoint), with the anticipation that in the case of a failure, computation can be restarted from a previously saved checkpoint. Coordinated checkpointing is a popular approach for its ease of implementation. Specifically, all processes coordinate with one another to produce individual states that satisfy the "happens before" communication relationship [Chandy and Ramamoorthy, 1972], which is proved to provide a consistent global state. The major benefit of coordinated checkpointing stems from its simplicity and ease of implementation. Its major drawback, however, is the lack of scalability, as it requires global coordination [Elnozahy and Plank, 2004; Riesen et al., 2010].

In uncoordinated checkpointing, processes checkpoint their states independently and postpone creating a globally consistent view until the recovery phase. The major advantage is the reduced overhead during fault free operation. However, the scheme requires that each process maintains multiple checkpoints and can also suffer the well-known domino effect [Randell, 1975; Alvisi et al., 1999; Helary and et. al., 1997]. One hybrid approach, known as communication induced checkpointing, aims at reducing coordination overhead [Alvisi et al., 1999]. The approach, however, may cause processes to store useless states. To address this shortcoming, "forced checkpoints" have been proposed [Helary and et. al., 1997]. This approach, however, may lead to unpredictable checkpointing rates. Although well-explored, uncoordinated checkpointing has not been widely adopted in HPC environments for its complexities.

One of the largest overheads in any checkpointing process is the time necessary to write the checkpointing to stable storage. Incremental checkpointing attempts to address this

by only writing the changes since previous checkpoint [Agarwal and et. al.; Elnozahy and Zwaenepoel, 1992; Li et al., 1994]. This can be achieved using dirty-bit page flags [Plank and Li, 1994; Elnozahy and Zwaenepoel, 1992]. Hash based incremental checkpointing, on the other hand, makes use of hashes to detect changes [chang Nam et al., 1997; Agarwal and et. al.]. Another proposed scheme, known as in-memory checkpointing, minimizes the overhead of disk access [Zheng and et. al., 2004; Zheng et al., 2012]. The main concern of these techniques is the increase in memory requirement to support the simultaneous execution of the checkpointing and the application. It has been suggested that nodes in extreme-scale systems should be configured with fast local storage [Ahern and et. al., 2011]. Multi-level checkpointing , which consists of writing checkpoints to multiple storage targets, can benefit from such a strategy [Moody et al., 2010]. This, however, may lead to increased failure rates of individual nodes and complicate the checkpoint writing process.

Process replication, or state machine replication, has long been used for reliability and availability in distributed and mission critical systems [Schneider, 1990]. Although it is initially rejected in HPC communities, replication has recently been proposed to address the deficiencies of checkpointing for upcoming extreme-scale systems [Cappello, 2009; Engelmann and Böhm, 2011]. Full and partial process replication have also been studied to augment existing checkpointing techniques, and to detect and correct silent data corruption [Stearley and et. al., 2012; Elliott and et. al.; Ferreira et al.; Fiala and et. al., 2012]. Our approach is largely different from classical process replication in that we dynamically configure the execution rates of main and shadow processes, so that less resource/energy is required while reliability is still assured.

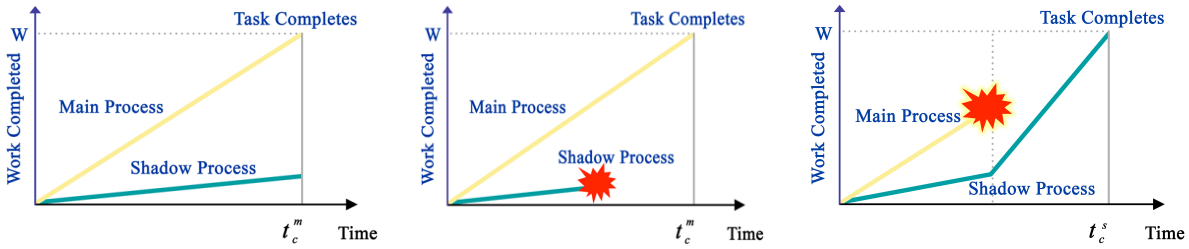
Replication with dynamic execution rate is also explored in Simultaneous and Redundantly Threaded (SRT) processor whereby one leading thread is running ahead of trailing threads [Reinhardt and et. al., 2000]. However, the focus of [Reinhardt and et. al., 2000] is on transient faults within CPU while we aim at tolerating both permanent and transient faults across all system components.

3.0 LAZY SHADOWING: A NOVEL FAULT-TOLERANT COMPUTATIONAL MODEL

Current fault tolerance approaches rely exclusively on either time or hardware redundancy to hide failures from being seen by users. Rollback recovery, which exploits time redundancy, requires full or partial re-execution when failure occurs. Such an approach can incur a significant delay, and high power costs due to extended execution time. On the other hand, Process Replication relies on hardware redundancy and executes multiple instances of the same task in parallel to guarantee completion with minimal delay. This solution, however, requires a significant increase in hardware resources and increases the power consumption proportionally.

It is without doubt that our understanding of how to build reliable systems out of unreliable components has led the development of robust and fairly reliable large-scale software and networking systems. The inherent instability of extreme-scale distributed systems of the future in terms of the envisioned high-rate and diversity of faults, however, calls for a reconsideration of the fault tolerance problem as a whole.

My proposed approaches to resiliency go beyond adapting or optimizing well known and proven techniques, and explore radically different methodologies to fault tolerance that scale to extreme-scale computing infrastructures. The proposed solutions differ in the type of faults they manage, their design, and the fault tolerance protocols they use. It is not just a scale up of "point" solutions, but an exploration of innovative and scalable fault tolerance frameworks. When integrated, it will lead to efficient solutions for a "tunable" resiliency that takes into consideration the nature of the data and the requirements of the application.



(a) No Failure (b) Shadow Process Failure (c) Main Process Failure

Figure 3.1: Shadow replication for a single task and single replica

3.1 COMPUTATIONAL MODEL

The basic tenet of Shadow Replication is to associate with each main process a suite of "shadows" whose size depends on the "criticality" of the application and its performance requirements, as defined by the SLA.

Assuming the fail-stop fault model, where a processor stops execution once a fault occurs and failure can be detected by other processors [Gärtner, 1999; Cristian, 1991], we define the Shadow Replication fault-tolerance model as follows:

- A main process, $P_m(W,\ \sigma_m)$, whose responsibility is to executes a task of size W at a speed of σ_m;
- A suite of shadow processes, $P_s(W,\ \sigma_b^s,\ \sigma_a^s)$ $(1 \leq s \leq \mathcal{S})$, where $\mathcal{S}$ is the size of the suite. The shadows execute on separate computing nodes. Each shadow process is associated with two execution speeds. All shadows start execution simultaneously with the main process at speed σ_b^s $(1 \leq s \leq \mathcal{S})$. Upon failure of the main process, all shadows switch their executions to σ_a^s, with one shadow being designated as the new main process. This process continues until completion of the task.

To illustrate the behavior of Shadow Replication, we limit the number of shadows to a single process and consider the scenarios depicted in Figure 3.1, assuming a single process failure. Figure 1(a) represents the case when neither the main nor the shadow fails. The

main process, executing at a higher speed, completes the task at time t_c^m. At this time, the shadow process, progressing at a lower speed, stops execution immediately. Figure 1(b) represents the case when the shadow fails. This failure, however, has no impact on the progress of the main process, which still completes the task at t_c^m. Figure 1(c) depicts the case when the main process fails while the shadow is in progress. After detecting the failure of the main process, the shadow begins execution at a higher speed, completing the task at time t_c^s. When possible, the shadow execution speed upon failure must be set so that t_c^s does not exceed t_c^m. Given that the failure rate of an individual node is much lower than the aggregate system failure, it is very likely that the main process will always complete its execution successfully, thereby achieving fault tolerance at a significantly reduced cost of energy consumed by the shadow.

A closer look at the model reveals that shadow replication is a generalization of traditional fault tolerance techniques, namely re-execution and traditional replication. If the SLA specification allows for flexible completion time, shadow replication would take advantage of the delay laxity to trade time redundancy for energy savings. It is clear, therefore, that for a large response time, Shadow Replication converges to re-execution, as the shadow remains idle during the execution of the main process and only starts execution upon failure. If the target response time is stringent, however, Shadow Replication converges to pure replication, as the shadow must execute simultaneously with the main at the same speed. The flexibility of the Shadow Replication model provides the basis for the design of a fault tolerance strategy that strikes a balance between task completion time and energy saving, thereby maximizing profit.

3.2 A USE CASE IN CLOUD COMPUTING

Cloud computing workload ranges from business applications and intelligence, to analytics and social networks mining and log analysis, to scientific applications in various fields of sciences and discovery. These applications exhibit different behaviors, in term of computation requirements and data access patterns. While some applications are compute-intensive,

others involve the processing of increasingly large amounts of data. The scope and scale of these applications are such that an instance of a job running one of these applications requires the sequential execution of multiple computing phases; each phase consists of thousands, if not millions, of tasks scheduled to execute in parallel and involves the processing of a very large amount of data [**??**]. This model is directly reflective of the *MapReduce* computational model, which is predominately used in Cloud Computing [**?**]. An instance of this model, is depicted in Figure 3.2.

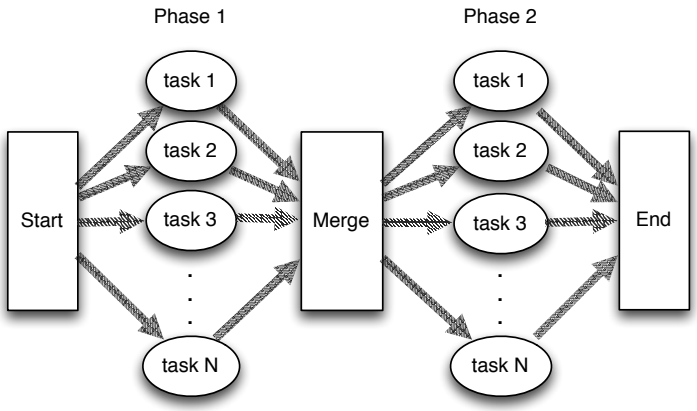


Figure 3.2: Cloud computing execution model with 2 phases.

Each task is mapped to one compute core and executes at a speed, σ. The partition of the job among tasks is such that each task processes a similar workload, W. Consequently, baring failures, tasks are expected to complete at about the same time. Therefore, the minimal response time of each task, when no failure occurs, is $t_{min} = \frac{W}{\sigma_{max}}$, where σ_{max} is the maximum speed. This is also the minimal response time of the entire phase.

As the number of tasks increases, however, the likelihood of a task failure during an execution of a given phase increases accordingly. This underscores the importance of an energy-efficient fault-tolerance model to mitigate the impact of a failing task on the overall delay of the execution phase. Lazy Shadowing is a perfect match for the needs. We can easily apply Lazy Shadowing by issuing one main and shadow pair for each task, and the execution can be performed phase by phase, just as previously.

3.3 REWARD-BASED OPTIMAL LAZY SHADOWING

In this section, we describe a profit-based optimization framework for the cloud-computing execution model previous described. Using this framework we compute profit-optimized execution speeds by optimizing the following objective function:

$$\begin{aligned} &\max_{\sigma_m,\sigma_b,\sigma_a} E[profit] \\ &s.t. 0 \leq \sigma_m \leq \sigma_{max} \\ &0 \leq \sigma_b \leq \sigma_m \\ &0 \leq \sigma_a \leq \sigma_{max} \end{aligned} \tag{3.1}$$

We assume that processor speeds are continuous and use nonlinear optimization techniques to solve the above optimization problem.

In order to earn profit, service providers must either increase income or decrease expenditure. We take both factors into consideration for the purpose of maximizing profit while meeting customer's requirements. In our model, we set the expected profit to be expected income minus expected expense.

$$E[\text{profit}] = E[\text{income}] - E[\text{expense}] \tag{3.2}$$

3.3.1 Reward Model

The cloud computing SLA can be diverse and complex. To focus on the profit and reliability aspects of the SLA, we define the reward model based on job completion time. Platform as a Service (PaaS) companies will continue to become more popular causing an increase in SLAs using job completion time as their performance metric. We are already seeing this appear in web-based remote procedure calls and data analytic requests.

As depicted in Figure 3.3, customers expect that their job deployed on cloud finishes by a mean response time t_{R_1}. As a return, the provider earns a certain amount of reward, denoted by R, for satisfying customer's requirements. However, if the job cannot be completed by the expected response time, the provider loses a fraction of R proportional to the delay incurred.

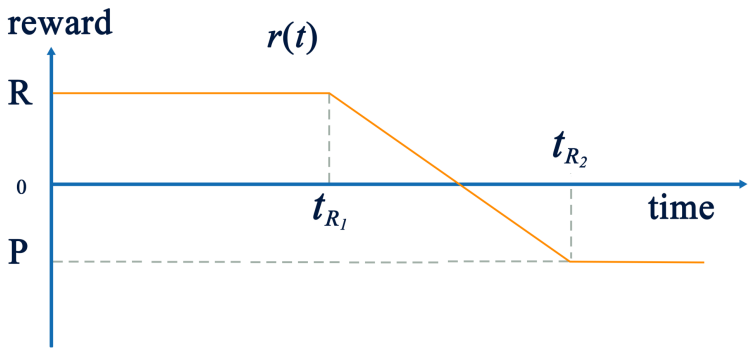


Figure 3.3: A reward function

For large delay, the profit loss may translate into a penalty that the CSP must pay to the customer. In this model, the maximum penalty P is paid if the delay reaches or exceeds t_{R_2}. The four parameters, R, P, t_{R_1} and t_{R_2}, completely define the reward model.

There are two facts that the service provider must take into account when negotiating the terms of the SLA. The first is the response time of the main process assuming no failure (Figure 1(a) and Figure 1(b)). This results in the following completion time:

$$t_c^m = W/\sigma_m \tag{3.3}$$

If the main process fails (shown in Figure 1(c)), the task completion time by shadow process is the time of the failure, t_f, plus the time necessary to complete the remaining work.

$$t_c^s = t_f + \frac{W - t_f \times \sigma_b}{\sigma_a} \tag{3.4}$$

This reward model is flexible and extensible; it is not restricted to the form shown in Figure 3.3. In particular, the decrease may be linear, concave, or convex and the penalty can extend to infinity. This model can further be extended to take into consideration both the short-term income and long-term reputation of the service provider [**?**].

3.3.2 Failure Model

Failure can occur at any point during the execution of the main or shadow process. Our assumption is that at most one failure occurs, therefore if the main process fails it is implied that the shadow will complete the task without failure. We can make this assumption because we know the failure of any one node is rare thus the failure of any two specific nodes is very unlikely.

We assume that two probability density functions, $f_m(t_f)$ and $f_s(t_f)$, exist which express the probabilities of the main and shadow process failing at time t_f separately. The model does not assume a specific distribution. However, in the remainder of this paper we use an exponential probability density function, $f_m(t_f) = f_s(t_f) = \lambda e^{-\lambda t_f}$, of which the mean time between failure (MTBF) is $\frac{1}{\lambda}$.

3.3.3 Power and Energy Models

Dynamic voltage and frequency scaling (DVFS) has been widely exploited as a technique to reduce CPU dynamic power [**??**]. It is well known that one can reduce the dynamic CPU power consumption at least quadratically by reducing the execution speed linearly. The dynamic CPU power consumption of a computing node executing at speed σ is given by the function $p_d(\sigma) = \sigma^n$ where $n \geq 2$.

In addition to the dynamic power, CPU leakage and other components (memory, disk, network etc.) all contribute to static power consumption, which is independent of the CPU speed. In this paper we define static power as a fixed fraction of the node power consumed when executing at maximum speed, referred to as ρ. Hence node power consumption is expressed as $p(\sigma) = \rho \times \sigma_{max}^n + (1-\rho) \times \sigma^n$. When the execution speed is zero the machine is in a sleep state, powered off or not assigned as a resource; therefore it will not be consuming any power, static or dynamic. Throughout this paper we assume that dynamic power is cubic in relation to speed [**??**], therefore the overall system power when executing at speed

σ is defined as:

$$p(\sigma) = \begin{cases} \rho\sigma_{max}^3 + (1-\rho)\sigma^3 & \text{if } \sigma > 0 \\ 0 & \text{if } \sigma = 0 \end{cases} \tag{3.5}$$

Using the power model given by 3.5, the energy consumed by a process executing at speed σ during an interval T is given by

$$E(\sigma, T) = p(\sigma) \times T \tag{3.6}$$

Corresponding to 3.1, there are three failure cases to consider: main and shadow both succeed, shadow fails and main fails. As described earlier, the case of both the main and shadow failing is very rare and will be ignored. The expected energy consumption for a single task is then the weighted average of the expected energy consumption in the three cases.

First consider the case where no failure occurs and the main process successfully completes the task at time t_c^m, corresponding to 1(a).

$$\begin{aligned} E_1 =& (1 - \int_0^{t_c^m} f_m(t)dt) \times (1 - \int_0^{t_c^m} f_s(t)dt) \times \\ & (E(\sigma_m, t_c^m) + E(\sigma_b, t_c^m)) \end{aligned} \tag{3.7}$$

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

Next, consider the case where the shadow process fails at some point before the main process successfully completes the task, corresponding to 1(b).

$$\begin{aligned} E_2 =& (1 - \int_0^{t_c^m} f_m(t)dt) \times \\ & \int_0^{t_c^m} (E(\sigma_m, t_c^m) + E(\sigma_b, t)) \times f_s(t)dt \end{aligned} \tag{3.8}$$

The first factor is the probability that the main process does not fail, and the probability of shadow fails is included in the second factor which also contains the energy consumption

since it depends on the shadow failure time. Energy consumption comes from the main process until the completion of the task, and the shadow process before its failure.

The one remaining case to consider is when the main process fails and the shadow process must continue to process until the task completes, corresponding to Figure 1(c).

$$\begin{aligned} E_3 =&(1 - \int_0^{t_c^m} f_s(t)dt) \times \int_0^{t_c^m} (E(\sigma_m, t)+ \\ &E(\sigma_b, t) + E(\sigma_a, t_c^s - t))f_m(t)dt \end{aligned} \tag{3.9}$$

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

The three equations above describe the expected energy consumption by a pair of main and shadow processes for completing a task under different situations. However, under our system model it might be the case that those processes that finish early will wait idly and consume static power if failure delays one task. If it is the case that processes must wait for all tasks to complete, then this energy needs to be accounted for in our model. The probability of this is the probability that at least one main process fails, referred to as the system level failure probability.

$$P_f = 1 - (1 - \int_0^{t_c^m} f_m(t)dt)^N \tag{3.10}$$

Hence, we have the fourth equation corresponding to the energy consumed while waiting in idle.

$$\begin{aligned} E_4 =&(1 - \int_0^{t_c^m} f_m(t)dt) \times (1 - \int_0^{t_c^m} f_s(t)dt)\times \\ &2P_f \times E(0, t_c^j - t_c^m) + \int_0^{t_c^m} f_s(t)dt\times \\ &(1 - \int_0^{t_c^m} f_m(t)dt) \times P_f \times E(0, t_c^j - t_c^m) \end{aligned} \tag{3.11}$$

Corresponding to the first case, neither main process nor shadow process fails, but both of them have to wait in idle from task completion time t_c^m to the last task's completion (by a shadow process) with probability P_f. Under the second case, only the main process has to wait if some other task is delayed since its shadow process has already failed. These two aspects are accounted in the first and last two lines in E_4 separately. We use the expected shadow completion time t_c^j as an approximation of the latest task completion time which is also the job completion time.

By summing these four parts and then multiplying it by N we will have the expected energy consumed by Shadow Replication for completing a job of N tasks.

$$E[\text{energy}] = N \times (E_1 + E_2 + E_3 + E_4) \tag{3.12}$$

3.3.4 Income and Expense Models

The income is the reward paid by customer for the cloud computing services that they utilize. It depends on the reward function $r(t)$, depicted in 3.3, and the actual job completion time. Therefore, the income should be either $r(t_c^m)$, if all main processes can complete without failure, or $r^*(t_c^s)$ otherwise. It is worth noting that the reward in case of failure should be calculated based on the last completed task, which we approximate by calculating the expected time of completion allowing us to derive the expected reward, i.e. $r^*(t_c^s) = \frac{\int_0^{t_c^m} r(t_c^s) \times f_m(t) dt}{\int_0^{t_c^m} f_m(t) dt}$. Therefore the income is estimated by the following equation.

$$E[\text{income}] = (1 - P_f) \times r(t_c^m) + P_f \times r^*(t_c^s) \tag{3.13}$$

The first part is the reward earned by the main process times the probability that all main processes would complete tasks without failure. If at least one main process fails, that task would have to be completed by a shadow process. As a result, the second part is the reward earned by shadow process times the system level failure probability.

If C is the charge expressed as dollars per unit of energy consumption (e.g. kilowatt hour), then the expected expenditure would be C times the expected energy consumption for all N tasks:

$$E[\text{expense}] = C \times E[\text{energy}] \tag{3.14}$$

However, the expenditure of running the cloud computing service is more than just energy, and must includes hardware, maintenance, and human labor. These costs can be accounted for by amortizing these costs into the static power factor, ρ. Because previous studies have suggested [**??**] that energy will become a dominate factor we decided to focus on this challenge and leave other aspects to future work.

Based on the above formalization of the optimization problem, the MATLAB Optimization Toolbox [**?**] was used to solve the resulting nonlinear optimization problem.

4.0 APPLYING LAZY SHADOWING TO EXTREME-SCALE SYSTEMS

5.0 LSMPI: IMPLEMENTATION OF LAZY SHADOWING IN MPI

6.0 SMART SHADOWING

7.0 TIMELINE OF PROPOSED WORK

Table 7.1: Timeline of Proposed Work.

Date	**Content**	**Deliverable results**
Jan - Feb	Explore restoring in approximate computing in Section **??** of Chapter **??**	Pin-based framework for restoring approximation
Mar - May	Integrate restoring with information leakage in Section **??** of Chapter **??**	Experimental data of memory performance and security
Jun - Sep	Study restoring in Hybrid Memory Cube (HMC) in Section **??** of Chapter **??**	Modified simulator of temperature effect of restoring in HMC
Jul - Oct	Thesis writing	Thesis ready for defense
Oct - Dec	Thesis revising	Completed thesis

The proposed works will be undertaken as shown in the Table 7.1. I will start the effort with task (1) to develop Pin-based framework for approximate computing of restoring. This task involves program annotation, chip generation, QoS evaluation, and conventional performance simulation, etc. While multiple complicated subtasks are there, this task has been partially finished, and will not take much time to complete. Afterwards, I'll move to task (2) to study information leakage in restoring scenario, and this task is partially on basis of the previous approximation work. With the completion of task (2), the overall goal of exploring DRAM restoring in application level have been reached, and then I'll start the study restoring's temperature effect in HMC, i.e., task (3). The general infrastructure can be borrowed from my previous HMC work [**?**]. This task might be performed concurrently with other jobs, and thus might take more time to finish. At the end of task (3), the holistic exploration of DRAM restoring is considered finished, and thus I'll summarize all the tasks into my final thesis.

8.0 SUMMARY

DRAM technology scaling has reached a threshold where physical limitations exert unprecedented hurdles on cell behaviors. Without dedicated mitigations, memory is expected to suffer from serious performance loss, yield degradation and reliability decrease, which goes against the demanding of system designs and applications. Among the induced problems, restoring has been an long time neglected issue, and it is likely to impose great constraints to the scaling advancement. As a result, this thesis explores DRAM further scaling from restoring perspective.

Reduced cell dimensions and worsening process variation cause increasingly slow access and significantly more outliers falling beyond the specifications. To alleviate the influences on performance and yield, we propose to manage the timing constraints at fine chunk granularity, and thus more fast regions can be exposed to upper level. In addition, we devise the extra chunk remapping and dedicated rank formation to restrict the impacts of slow cells. However, the exposed fast regions can not be fully utilized for restoring oblivious page allocation; accordingly, to maximize performance gains, we move forward to profile the pages of the workloads, and deliberately allocate the hot pages to fast parts.

In addition, restoring can seek help from the correlated refresh operation, which periodically fully charge the cells. We propose to perform partial restore withe respect to the distance to next refresh; the closer to next refresh, the less needed charge and the earlier the restore operation can be terminated. For ease of implementation, we divide the refresh window into 4 sub ones, and apply an separate set of timings for each. Moreover, compared to refresh, restore contributes more critically to the overall performance, and hence we optimize the partial restore with refresh rate upgrading. More frequent refresh helps to lower the restoring objectives, but at a risk of raising energy consumption. As a compromise, we

selectively upgrade recent touched rows only.

As the most fundamental building block of computer systems, DRAM prolonged restoring operation will ultimately affect upper application level, and thus we further explore in extended scenarios, including approximate computing, information leakage and 3D stacked memory. With our full-stack exploration, we believe the scaling issues can be greatly alleviated.

BIBLIOGRAPHY

Agarwal, Saurabh and al. et. . Adaptive incremental checkpointing for massively parallel systems. In *ICS 04*, St. Malo, France.

Ahern, Sean and al. et. . Scientific discovery at the exascale, a report from the doe ascr 2011 workshop on exascale data management, analysis, and visualization, 2011.

Alvisi, L., Elnozahy E., Rao S., Husain S. A., and Mel A.de . An analysis of communication induced checkpointing. In *Fault-Tolerant Computing*, 1999. doi: 10.1109/FTCS.1999.781058.

Cappello, Franck. Fault tolerance in petascale/ exascale systems: Current knowledge, challenges and research opportunities. *IJHPCA*, 23(3):212–226, 2009.

Chandy, K.M. and Ramamoorthy C.V. Rollback and recovery strategies for computer programs. *Computers, IEEE Transactions on*, C-21(6):546–556, June 1972. ISSN 0018-9340. doi: 10.1109/TC.1972.5009007.

Nam, Hyochang , Kim Jong, Lee Sunggu, and Lee Sunggu. Probabilistic checkpointing. In *In Proceedings of Intl. Symposium on Fault-Tolerant Computing*, pages 153–160, 1997.

Cristian, Flavin. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78, February 1991. ISSN 0001-0782. doi: 10.1145/102792.102801.

Elliott, James and al. et. . Combining partial redundancy and checkpointing for HPC. In *ICDCS '12*, Washington, DC, US. ISBN 978-0-7695-4685-8. doi: 10.1109/ICDCS.2012.56.

Elnozahy, E. and al. et. . A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/568522.568525.

Elnozahy, Elmootazbellah and Zwaenepoel Willy. Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit. *TC*, 41:526–531, 1992.

Elnozahy, E.N. and Plank J.S. Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery. *DSC*, 1(2):97 – 108, april-june 2004. ISSN 1545-5971. doi: 10.1109/TDSC.2004.15.

Engelmann, Christian and Böhm Swen. Redundant execution of hpc applications with mr-mpi. In *PDCN*, pages 15–17, 2011.

Ferreira, Kurt, Stearley Jon, Laros James H., III, Oldfield Ron, Pedretti Kevin, Brightwell Ron, Riesen Rolf, Bridges Patrick G., and Arnold Dorian. 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. ISBN 978-1-4503-0771-0. doi: 10.1145/2063384.2063443.

Fiala, David and al. et. . Detection and correction of silent data corruption for large-scale high-performance computing. SC, Los Alamitos, CA, USA, 2012. ISBN 978-1-4673-0804-5.

Gärtner, Felix C. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26, March 1999. ISSN 0360-0300. doi: 10.1145/311531.311532.

Helary, J and al. et. . Preventing useless checkpoints in distributed computations. In *RDS*, 1997. doi: 10.1109/RELDIS.1997.632814.

Li, K., Naughton J. F., and Plank J. S. Low-latency, concurrent checkpointing for parallel programs. *IEEE Trans. Parallel Distrib. Syst.*, 5(8):874–879, August 1994. ISSN 1045-9219. doi: 10.1109/71.298215.

Moody, Adam, Bronevetsky Greg, Mohror Kathryn, and Supinski Bronis. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC*, pages 1–11, 2010. ISBN 978-1-4244-7559-9. doi: http://dx.doi.org/10.1109/SC.2010.18.

Plank, J.S. and Li Kai. Faster checkpointing with n+1 parity. In *Fault-Tolerant Computing*, pages 288–297, June 1994. doi: 10.1109/FTCS.1994.315631.

Randell, B. System structure for software fault tolerance. In *Proceedings of the international conference on Reliable software*, New York, NY, USA, 1975. ACM. doi: 10.1145/800027.808467.

Reinhardt, Steven K and al. et. . *Transient fault detection via simultaneous multithreading*, volume 28. ACM, 2000.

Riesen, Rolf, Ferreira Kurt, Stearley Jon R., Oldfield Ron, III James H. Laros, Pedretti Kevin T., and Brightwell Ron. Redundant computing for exascale systems, December 2010.

Schneider, Fred B. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, December 1990. ISSN 0360-0300. doi: 10.1145/98163.98167.

Stearley, J. and al. et. . Does partial replication pay off? In *DSN-W*, pages 1–6, June 2012. doi: 10.1109/DSNW.2012.6264669.

Zheng, G., Ni Xiang, and Kal L. V. A scalable double in-memory checkpoint and restart scheme towards exascale. In *DSN-W*, pages 1–6, June 2012. doi: 10.1109/DSNW.2012.6264677.

Zheng, Gengbin and al. et. . FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In *Cluster Computing*, pages 93–103, 2004. doi: 10.1109/CLUSTR.2004.1392606.