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

by

Xiaolong Cui

B.E. in Computer Science, Xi'an Jiaotong University, 2012

M.S. in Computer Science, University of Pittsburgh, 2017

Submitted to the Graduate Faculty of

the School of Computing and Information in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2017

UNIVERSITY OF PITTSBURGH

SCHOOL OF COMPUTING AND INFORMATION

This dissertation was presented

by

Xiaolong Cui

It was defended on

September 19, 2017

and approved by

Dr. Taieb Znati, Departmental of Computer Science, 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, Departmental of Computer Science, University of Pittsburgh,

Co-advisor: 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, 2017

Two major trends in large-scale computing systems are the rapid growth in High Performance Computing (HPC) with in particular an international exascale initiative, and the dramatic expansion of Cloud infrastructure accompanied by the Big Data trends. To satisfy the continuous demand for increasing computing capacity, future extreme-scale systems will embrace a multi-fold increase in the number of computing, storage, and communication components, in order to support an unprecedented level of parallelism. Despite the power and economies of scale, making the transition to extreme-scale poses numerous unavoidable scientific and technological challenges.

As the demand for computing power continues to increase, both HPC community and Cloud service providers 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 datacenters, for both private usage and public services. However, making the transition to extreme-scale poses numerous unavoidable scientific and technological challenges.

This thesis aims at simultaneously solving two major challenges, i.e., power consumption and fault tolerance, for future extreme-scale computing systems. We study a novel computational model that is power-aware and fault-tolerant. Different techniques have been explored to embody this model. Accordingly, precise analytical models and optimization framework have been developed to quantify and optimize the performance, respectively.

In this document, I summarize my progress in the development of the Lazy Shadowing concept and 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 adaptivity 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 and the viability of partial shadowing.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

For my beloved families.

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, the computing power attributes to the massive amount of parallelism, which is enabled by the massive amount of CPU cores, memory modules, communication devices, storage components, etc.

Since CPU frequency flattened 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 more 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 [2, 37].

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 economies of scale. For example, 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. 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 computing, two of the biggest challenges will be system resilience and power consumption, both being direct consequences of the dramatic increase in the number of system components [4, 53].

Regardless of the reliability of individual component, the system level failure rate will continue to increase as the number of components increases, possibly by several orders of magnitude. 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 [6]. Without an efficient fault tolerance mechanism, 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 [4]. 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 equipped 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 [24, 32, 11]. Although checkpointing is the most widely used technique in today's HPC systems, it may not scale to future extreme-scale systems [27, 22, 45]. Given the anticipated increase in system level failure rates and the time to checkpoint large-scale compute-intensive and data-intensive applications, the time required to periodically checkpoint an application and restart its execution will approach the system's MTBF [8]. 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 [5, 57, 27]. 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.

Based on above analysis, neither of the two approaches is 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 [9, 58]. The trade-off between fault free operation and optimal power consumption has been explored in the literature [38, 41]. 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.

To this end, Mills, Znati, and Melhem have proposed a novel computational model, referred to as Shadow Replication, as a power-aware approach to achieve high-levels of resilience through forward progress [40, 42, 39]. Based on Dynamic Voltage and Frequency Scaling (DVFS) [46, 34, 35], Mills studied the computational model and its performance in terms of completion time and energy consumption in HPC systems. Through the use of analytical models, simulations, and experimentation, Mills demonstrated that Shadow Replication can achieve resilience more efficiently than both checkpointing and traditional replication when power is limited. However, in Mills' work Shadow Replication is limited to the use of DVFS, which has been shown to have multiple issues that question its viability [26, 33, 9, 58]. In addition, Mills' study is limited to HPC systems and focuses exclusively on minimizing energy consumption with constraints on time to completion. In contrast, QoS requirements for various computing systems can be expressed in multiple dimensions that go beyond time and energy. At the same time, an implementation is needed to verify the computational model both with and without failures.

To address the above limitations, this thesis builds on the computational model of Shadow Replication, and seeks to simultaneously address the power and resilience challenges for future extreme-scale systems while guaranteeing system efficiency and application QoS. Specifically, this thesis tries to answer 4 questions: 1) is Shadow Replication general enough to achieve objectives beyond time to completion; 2) how to enable Shadow Replication when DVFS is not viable, and ensure forward progress in failure-prone, extreme-scale systems; 3) is the computational model realistic in real environments; and 4) how to make the computational model reflective of the propensity of the processing elements to failures and adaptive to different environments and requirements. With these questions in mind, I have studied

different techniques to embody and augment the model, and developed analytical frameworks for different objectives in the Cloud and HPC environments [15, 14, 16]. To complete my thesis, I propose to extend the study in the following two aspects. Firstly, I propose to implement a prototype in the context of Message Passing Interface (MPI), to validate the computational model as well as measure its performance in real environment. Secondly, I propose to study "smart shadowing" which adapts to the system configuration, application characteristics, and QoS requirement. In summary, my thesis will consist of the following main components.

1.2.1 Reward-based optimal Shadow Replication (completed)

Shadow Replication is a flexible computational model that can achieve multi-dimensional QoS requirements. The major challenge 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, power consumption, or their combinations. In this work we focus on the Service Level Agreement (SLA) requirements in the Cloud and develop a reward-based analytical framework, in order to derive the optimal execution rates for maximizing reward and minimizing energy costs under strict completion time constraints [15, 14].

1.2.2 Lazy Shadowing (completed)

Enabling Shadow Replication for resiliency in extreme-scale computing brings about a number of challenges and design decisions, 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 devise novel ideas of shadow collocation and shadow leaping, and integrate them with Shadow Replication to form a more efficient and scalable paradigm that we call Lazy Shadowing [16].

1.2.3 lsMPI: an implementation in MPI (in progress)

Though Lazy Shadowing has been evaluated analytically, a real implementation is necessary for validation and performance measurement in real systems. I am implementing a prototype of Lazy Shadowing as a runtime for Message Passing Interface (MPI). The runtime will spawn the shadow processes at initialization phase, manage the coordination between main and shadow processes, and guarantee order and consistency for messages and non-deterministic events. With this implementation, we will perform thorough experiments to measure 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., 19% of the nodes account for 92% of the machine check errors on Blue Waters [19]. The reason is complicated and may attribute to the manufacture process, heterogeneous architecture, environment factors (e.g. temperature), and/or workloads. Under differernt failure distribution assumptions, 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

When completed, this thesis will make the following contributions:

- A reward-based framework for Shadow Replication to satisfy SLA requirements as well as maximize profit in Cloud Computing
- Study of Lazy Shadowing as an enhanced model for future extreme-scale systems
- A fully functional implementation of Lazy Shadowing for Message Passing Interface

- Exploration of Lazy Shadowing's adaptivity to different environments, workloads, and QoS requirements.

1.4 OUTLINE

The rest of this proposal is organized as follow: Chapter 2 reviews existing fault tolerance techniques in large-scale distributed systems, and Chapter 3 introduces the Shadow Replication computational model. In Chapter 4 we build a reward-based optimization framework for Shadow Replication in the Cloud environment. In Chapter 5, we introduce Lazy Shadowing which enhances Shadow Replication for extreme-scale systems. Implementation issues are discussed in Chapter 6. Adaptivity and smart shadowing are discussed in Chapter **??**. Chapter **??** concludes the proposal.

2.0 BACKGROUND

Rollback recovery is the dominant mechanism to achieve fault tolerance in current HPC environments [18, 24, 20]. 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 [10], 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 [22, 50].

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 [48, 3, 31]. One hybrid approach, known as communication induced checkpointing, aims at reducing coordination overhead [3]. The approach, however, may cause processes to store useless states. To address this shortcoming, "forced checkpoints" have been proposed [31]. This approach, however, may lead to unpredictable checkpointing rates. Although well-explored, uncoordinated checkpointing has not been widely adopted in HPC environments due to its complexities and its dependency on applications [30].

One of the largest overheads in any checkpointing process is the time necessary to write the checkpoints to stable storage. Incremental checkpointing attempts to address this by only writing the changes since previous checkpoint [1, 23, 36]. This can be achieved using dirty-bit page flags [47, 23]. Hash based incremental checkpointing, on the other hand, makes

use of hashes to detect changes [12, 1]. Another proposed scheme, known as in-memory checkpointing, minimizes the overhead of disk access [60, 59]. 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 [2]. Multi-level checkpointing , which consists of writing checkpoints to multiple storage targets, can benefit from such a strategy [43]. 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 [52, 54, 61]. Although it was initially rejected in HPC communities, replication has recently been proposed to address the deficiencies of checkpointing for upcoming extreme-scale systems [7, 25]. Full and partial process replication have also been studied to augment existing checkpointing techniques, and to detect and correct silent data corruption [55, 21, 27, 28, 44]. 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 [49]. However, the focus of [49] is on transient faults within CPU while we aim at tolerating both permanent and transient faults across all system components.

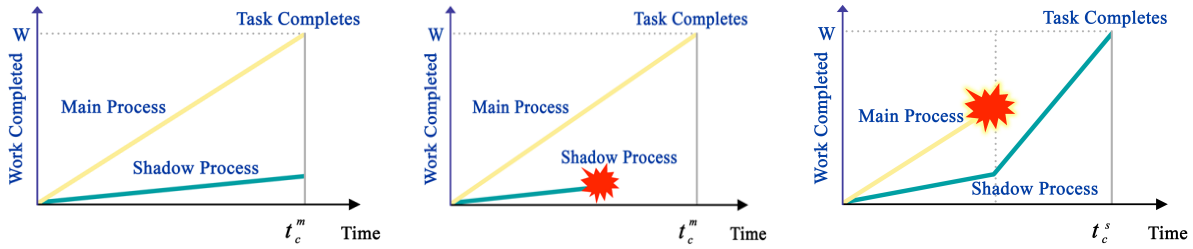
3.0 SHADOW REPLICATION

Shadow Replication is a novel computational model that goes beyond adapting or optimizing well known and proven techniques, and explores radically different methodologies to fault tolerance [40, 42, 39]. 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. Each shadow process is an exact replica of the original main process, and a consistency protocol is needed to assure that the main and the shadow are consistent. When possible, the shadow executes at a reduced rate to save power. Shadow Replication achieves power efficiency under QoS requirements by dynamically responding to failures.

Assuming the fail-stop fault model, where a processor stops execution once a fault occurs and failure can be detected by other processors [29, 13], the Shadow Replication fault-tolerance model can be described 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.

Assuming a single process failure, Figure 1 illustrates the behavior of Shadow Replication with one shadow per task. If the main process does not fail, it will complete the task ahead



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

Figure 1: Shadow Replication for a single task and single replica

of its shadow at t_c^m. At this time, we terminate the shadow immediately to save power. If the shadow fails before the main completes, the failure has no impact on the progress of the main. If the main fails, however, the shadow switches to a higher speed and completes the task at time t_c^s. 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 process replication. If it allows for flexible completion 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 process 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.

Based on DVFS, Mills studied the computational model in HPC systems and demonstrated that Shadow Replication can achieve resilience more efficiently than both checkpointing and process replication when power is limited [40, 42, 39]. I am going to extend the work to achieve better flexibility, adaptivity, and scalability.

4.0 REWARD-BASED OPTIMAL SHADOW REPLICATION

Cloud Computing has emerged as an attractive platform for diverse compute- and data-intensive applications, as it allows for low-entry costs, on demand resource provisioning, and reduced cost of maintenance [56]. As the demand for cloud computing accelerates, cloud service providers (CSPs) will be faced with the need to expand their underlying infrastructure to ensure the expected levels of performance and cost-effectiveness.

Two direct implications of large-scale datacenters are increased energy costs, which increase the operating expenditure, and service failures, which subject the CSP to a loss of revenue. Therefore, Service Level Agreement (SLA) becomes a critical aspect for a sustainable cloud computing business. To understand the question of how fault tolerance might impact power consumption and ultimately the expected profit of CSPs, we study the application of Shadow Replication for satisfying SLA in cloud computing.

4.1 OPTIMIZATION FRAMEWORK

We develop a reward model to abstract SLA terms in Cloud Computing. To maximize the expected profit for CSPs, we build a reward-based optimization framework:

$$\begin{aligned} &\max_{\sigma_m,\sigma_b,\sigma_a} E[profit] \\ &s.t. 0 \leq \sigma_m \leq \sigma_{max} \\ &0 \leq \sigma_b \leq \sigma_m \\ &0 \leq \sigma_a \leq \sigma_{max} \end{aligned} \tag{4.1}$$

To express the expected profit ($E[profit]$) considering failures, we further develop models for process failure distribution, power consumption, and service expenditure including energy cost. Assuming that processor speeds are continuous, we use nonlinear optimization techniques to solve the above optimization problem. Using this framework, we compute profit-optimized execution speeds for Shadow Replication. Please refer to [15] for more details.

Unlike traditional replication, Shadow Replication is dependent upon failure detection, enabling the replica to increase its execution speed upon failure and maintain the targeted response time thus maximizing profit. While this is the case in many computing environments, there are cases where failure detection may not be possible. To address this limitation, we propose profit-aware stretched replication, whereby both the main process and the shadow execute independently at stretched speeds to meet the expected response time, without the need for failure detection. In profit-aware stretched replication both the main and shadow execute at speed σ_r, found by optimizing the profit model.

4.2 PERFORMANCE EVALUATION

This section evaluates the expected profit of each of the fault tolerance methods using above optimization framework. We study different system environment and identify 5 important parameters which affect the expected profit:

- Static power ratio ρ, which determines the portion of power that is unaffected by the execution speed.
- SLA - The amount of reward, penalty and the required response times.
- N - The total number of tasks.
- MTBF - The reliability of an individual node.
- Workload - The size, W, of each individual task.

Without loss of generality, we normalize σ_{max} to be 1, so that all the speeds can be expressed as a fraction of maximum speed. Accordingly, the task workload W is also adjusted

such that it is equal to the amount of time (in hours) required for a single task. In our baseline configuration we assume that the static power ratio is 0.5, the node MTBF is 5 years, the number of tasks is 100000, the task size is 1 hour, and the response time thresholds for maximal and minimal rewards are 1.3 hours and 2.6 hours respectively.

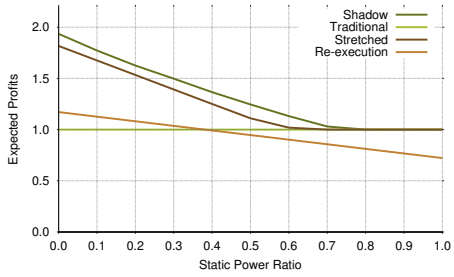
With various architectures and organizations, servers deployed at different data centers will have different characteristics in terms of power consumption. The static power ratio is used to abstract the amount of static power consumed versus dynamic power. The potential profit gains achievable by using profit-aware replication techniques decreases as static power increases, as is shown in Figure 2(a). The reason is that our profit-aware techniques rely upon the fact that one can reduce dynamic power by adjusting the execution speeds. Modern systems have a static power between 40%-70% and it is reasonable to suspect that this will continue to be the case. Within this target range of static power, Shadow Replication can achieve, on average, 19.3% more profit than traditional replication, 8.9% more than profit-aware stretched replication, and 28.8% more than re-execution.

Figure 2(b) shows the effect that targeted response time has upon the profitability of each fault tolerance method. Compared to traditional replication, all the other methods increase their profit as the targeted response time increases, this is expected because each of the other techniques can make use of increased laxity in time to increase profit. Re-execution is the most sensitive to the target response time since it fully relies upon time redundancy, showing that it should only be used when the targeted response time is *not* stringent. Again, Shadow Replication always achieves more profit than traditional replication and profit-aware stretched replication, and the profit gains are 52.8% and 39.0% on average.

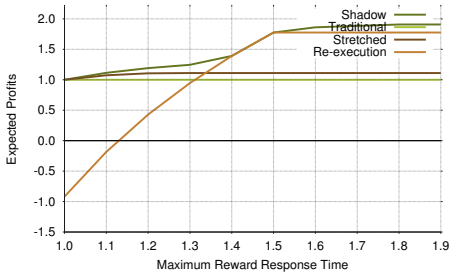
Figure 2(c) confirms that for small number of tasks re-execution is more profitable than replication. However, re-execution is not scalable as its profit decreases rapidly after N reaches 10000. At the same time, traditional replication and profit-aware stretched replication are not affected by the number of tasks because neither are affected by the system level failure rate. On average, Shadow Replication achieves 43.5%, 59.3%, and 18.4% more profits than profit-aware stretched replication, traditional replication and re-execution, respectively.

The ratio between task size and node MTBF represents the tasks vulnerability to failure, specifically it is an approximation of the probability that failure occurs during the execution

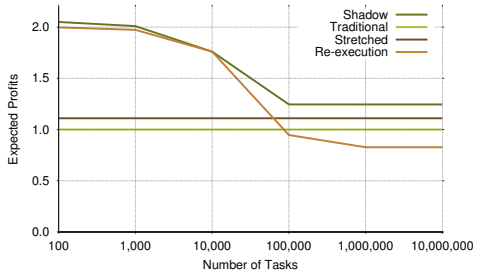
of the task. In our analysis we found that increasing task size will have the same effect as reducing node MTBF. Therefore, we analyze these together using the vulnerability to failure, allowing us to analyze a wider range of system parameters. As expected re-execution is desired when the vulnerability to failure is low. As always, Shadow Replication can adjust its execution strategy to maximize the profits, as shown in Figure 2(d).



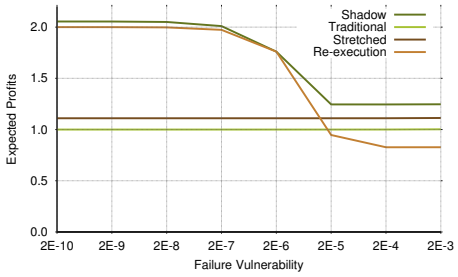
(a) Profit for different static power ratio. MTBF=5 years, N=100000, W=1 hour, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.



(b) Profit for different response time threshold. ρ=0.5, MTBF=5 years, N=100000, W=1 hour.



(c) Profit for different task size over MTBF. ρ=0.5, N=100000, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.



(d) Profit for different task size over MTBF. ρ=0.5, N=100000, t_{R_1}=1.3 hours, t_{R_2}=2.6 hours.

Figure 2: Profit comparison among fault tolerance methods.

Lastly, we evaluate the profit of each resilience technique using three different benchmark applications representing a wide range of application [51]: Business Intelligence, Bioinformatics and Recommendation System. While the results vary from application to application, they are aligned with above results and analysis. Please refer to [15] for more details.

4.3 SUMMARY

In this work we focus on the objective of satisfying SLA in Cloud Computing and demonstrate that Shadow Replication is capable of achieving multi-dimensional QoS goals. To assess the performance of the Shadow Replication, an analytical framework is developed and an extensive performance evaluation study is carried out. In this study, system properties that affect the profitability of fault tolerance methods, namely failure rate, targeted response time and static power, are identified. The failure rate is affected by the number of tasks and vulnerability of the task to failure. The targeted response time represents the clients' desired job completion time. Our performance evaluation shows that in all cases, Shadow Replication outperforms existing fault tolerance methods. Furthermore, shadow replication will converge to traditional replication when target response time is stringent, and to re-execution when target response time is relaxed or failure is unlikely.

5.0 LAZY SHADOWING

Enabling Lazy Shadowing for resiliency in extreme-scale computing brings about a number of challenges and design decisions, 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*, and integrate them with Shadow Replication to form a more efficient and scalable paradigm that we call Lazy Shadowing.

5.1 SHADOW COLLOCATION

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 [26, 33, 9, 58]. 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 each core, it simultaneously reduces the number of compute nodes required and reduces the power consumption.

To execute an application of M tasks, $N = M + S$ cores are required, where M is a multiple of S. Each main is allocated one core (referred to as *main core*), while $\alpha = M/S$ (referred to as *collocation ratio*) shadows are collocated on a core (*shadow core*). The N

cores are grouped into S sets, each of which we call a *shadowed set*. Each shadowed set contains α main cores and 1 shadow core. This is illustrated in Figure 3.

Collocation has an important ramification with respect to the resilience of the system. Specifically, one failure can be tolerated in each shadowed set. If a shadow core fails, all the shadows in the shadowed set will be lost without interrupting the execution of the mains. On the other hand, if a main core fails, the associated shadow will be promoted to a new main, and all the other collocated shadows will be terminated to speed up the new main. Consequently, a failure, either in main or shadow core, will result in losing all the shadows in the shadowed set, thereby losing the tolerance to any other failures. After the first failure, a shadowed set becomes *vulnerable*[1].

5.2 SHADOW LEAPING

As the shadows execute at a lower rate, failures will incur delay for recovery. This problem deteriorates as dependencies incurred by messages and synchronization barriers would prop-

[1]Rejuvenation techniques, such as restarting the lost shadows from the state of current mains on spare cores, can be used to eliminate vulnerability.

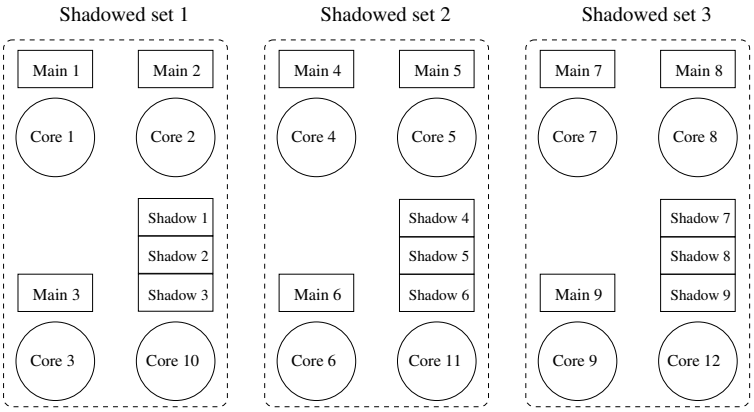


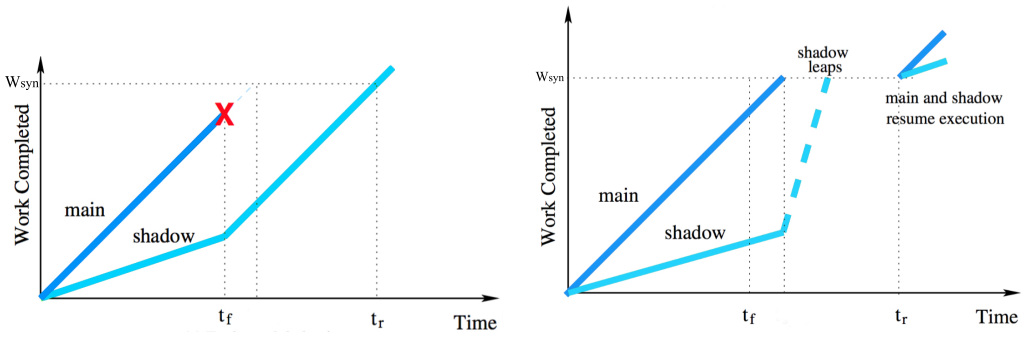
Figure 3: An example of collocation. $N = 12$, $M = 9$, $S = 3$.

agate the delay of one task to others. Fortunately, keeping the mains running at normal rate provides an opportunity for the shadows to benefit from the faster execution of their mains. By copying the state of each main to its shadow, which is similar to the process of storing a checkpoint in a buddy in [60], forward progress is achieved for the shadows with minimized time and energy. This technique, referred to as *shadow leaping*, effectively limits the distance between main and shadow in progress. As a result, the recovery time after a failure, which depends on the distance between the failing main and its shadow, is also reduced. More importantly, we opportunistically overlap shadow leaping with failure recovery to avoid extra overhead.

Assuming a failure occurrence at time t_f, Figure 4 shows the concept of shadow leaping. Upon failure of a main process, its associated shadow speeds up to minimize the impact of failure recovery on the other tasks' progress, as illustrated in Figure 4(a). At the same time, as shown in Figure 4(b), the remaining main processes continue execution until the barrier at W_{syn}, and then become idle until t_r. Shadow leaping opportunistically takes advantage of this idle time to *leap forward* the shadows, so that all processes, including shadows, can resume execution from a consistent point afterwards. Shadow leaping increases the shadow's rate of progress, at a minimal energy cost. Consequently, it reduces significantly the likelihood of a shadow falling excessively behind, thereby ensuring fast recovery while minimizing the total energy consumption.

5.3 PERFORMANCE EVALUATION

We develop analytical models to quantify the expected performance of Lazy Shadowing, as well as prove the bound on performance loss due to failures. Please refer to [16] for details. Careful analysis of the models leads us to identify several important factors that determine the performance. These factors can be classified into three categories, i.e., system, application, and algorithm. The system category includes static power ratio ρ ($\rho = p_s/p$), total number of cores N, and MTBF of each core; the application category is mainly the total workload, W; and shadowing ratio α in the algorithm category determines the number of



(a) Faulty task behavior.

(b) Non-faulty task behavior.

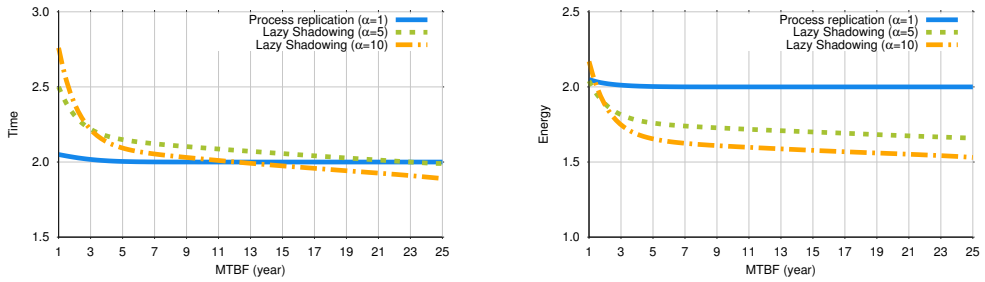
Figure 4: The illustration of shadow leaping.

main cores and shadow cores ($N = M + S$ and $\alpha = M/S$). In this section, we evaluate each performance metric of Lazy Shadowing, with the influence of each of the factors considered.

We compare with both process replication and checkpointing. The completion time with checkpointing is calculated with Daly's model [17] assuming 10 minutes for both checkpointing time and restart time. It is important to point out that we always assume the same number of cores available to use, so that process replication and Lazy Shadowing do not use extra cores for the replicas.

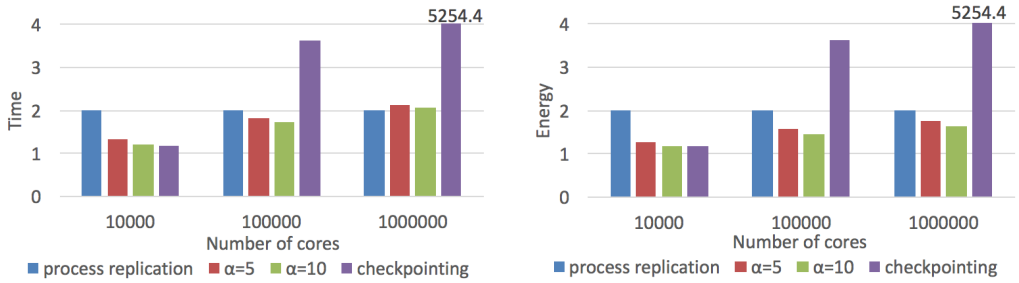
The first study uses $N = 1$ million cores, $W = 1$ million hours, and static power ratio $\rho = 0.5$. Our results show that at extreme-scale, the completion time and energy consumption of checkpointing are orders of magnitude larger than those of Lazy Shadowing and process replication. Thus, we choose not to plot a separate graph for checkpointing in the interest of space. Figure 5(a) reveals that the most time efficient choice largely depends on MTBF. When MTBF is high, Lazy Shadowing requires less time as more cores are used for main processes and less workload is assigned to each process. As MTBF decreases, process replication outperforms Lazy Shadowing as a result of the increased likelihood of rollback for Lazy Shadowing. In terms of energy consumption, Lazy Shadowing has a much larger advantage over process replication.

The system scale, measured in number of cores, has a direct impact on the failure rate



(a) Expected completion time (b) Expected energy consumption

Figure 5: Comparison of time and energy for different core level MTBF. $W = 10^6$ hours, $N = 10^6$, $\rho = 0.5$.



(a) Expected completion time (b) Expected energy consumption

Figure 6: Comparison of time and energy for different number of cores. $W = N$, MTBF=5 years, $\rho = 0.5$.

seen by the application. To study its impact, we vary N from 10,000 to 1,000,000 with W scaled proportionally, i.e., $W = N$. When MTBF is 5 years, the results are shown in Figure 6. Please note that the time and energy for checkpointing when $N = 1,000,000$ are beyond the scope of the figures, so we mark their values on top of their columns. When completion time is considered, Figure 6(a) clearly shows that each of the three fault tolerance alternatives has its own advantage. On the other hand, Lazy Shadowing wins for all system sizes when energy consumption is the objective.

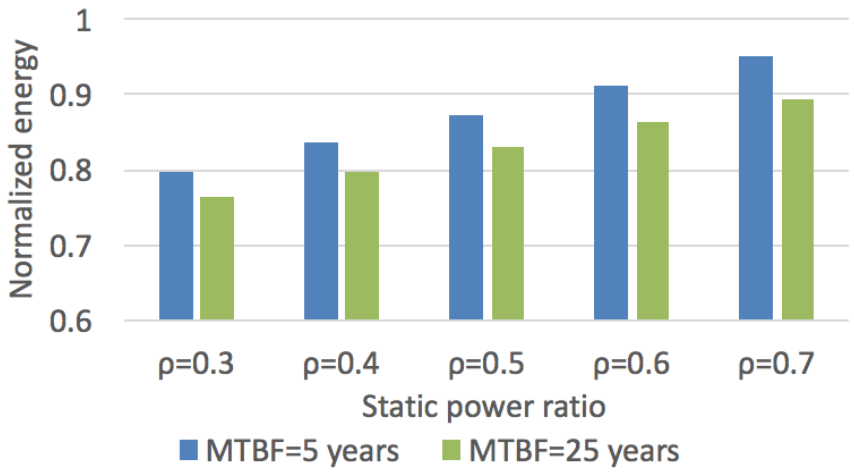


Figure 7: Impact of static power ratio on energy consumption. $W = 10^6$ hours, $N = 10^6$, α=5.

With various architectures and organizations, servers vary in terms of power consumption. The static power ratio ρ is used to abstract the amount of static power consumed versus dynamic power. Considering modern systems, we vary ρ from 0.3 to 0.7 and study its effect on the expected energy consumption. The results for Lazy Shadowing with $\alpha = 5$ are normalized to that of process replication and shown in Figure 7. Lazy Shadowing achieves more energy saving when the static power ratio is low, since it saves dynamic power but not static power.

5.4 SUMMARY

In this work, we present a comprehensive discussion of the techniques that enable Lazy Shadowing to achieve scalable resilience in future extreme-scale computing systems. In addition, we develop a series of analytical models to assess its performance in terms of reliability, completion time, and energy consumption. Through comparison with existing fault tolerance approaches, we identify the scenarios where each of the alternatives should

be chosen for best performance.

6.0 lsMPI: AN IMPLEMENTATION IN MPI

In a complex system like the ones we have today, the performance of Lazy Shadowing is subject to both the hardware configuration, such as failure detection and execution rate control, and software behavior, such as the amount of communication and synchronization. It's difficult for an analytical framework to precisely capture every details of Lazy Shadowing when it runs in a real environment. Therefore, a functional prototype is necessary to prove its validity as well as measure its actual performance.

We are implementing Lazy Shadowing as a library (lsMPI) for MPI, which is the de facto programming paradigm for HPC. Instead of a full-feature MPI implementation, the library is designed to be a separate layer between MPI and user application, and uses the MPI profiling hooks to intercept every MPI call. There are three benefits for this choice: 1) we can save tremendous time and efforts of rebuilding MPI from scratch; 2) we can take advantage of existing MPI performance optimization that numerous researches have spent years on; and 3) the library is portable across all MPI implementations. The library will spawn the shadow processes at the initialization phase, manage the coordination between main and shadow processes during execution, and guarantee order and consistency for messages and non-deterministic events. Once completed, users should be able to link to the library without any change to existing codes.

Same as rMPI [27], lsMPI uses the underlying Reliability, Availability and Serviceability (RAS) system to detect process failure. In this prototype system, we will emulate a RAS system at the user level, and use an event mechanism to inform lsMPI whenever RAS detects a failure. When initializing, each process installs a signal handler dedicated to failure detection and notification. To emulate a failure, RAS sends a signal to the process and triggers the handler, which forces the process to fail and uses out-of-band messages to notify other

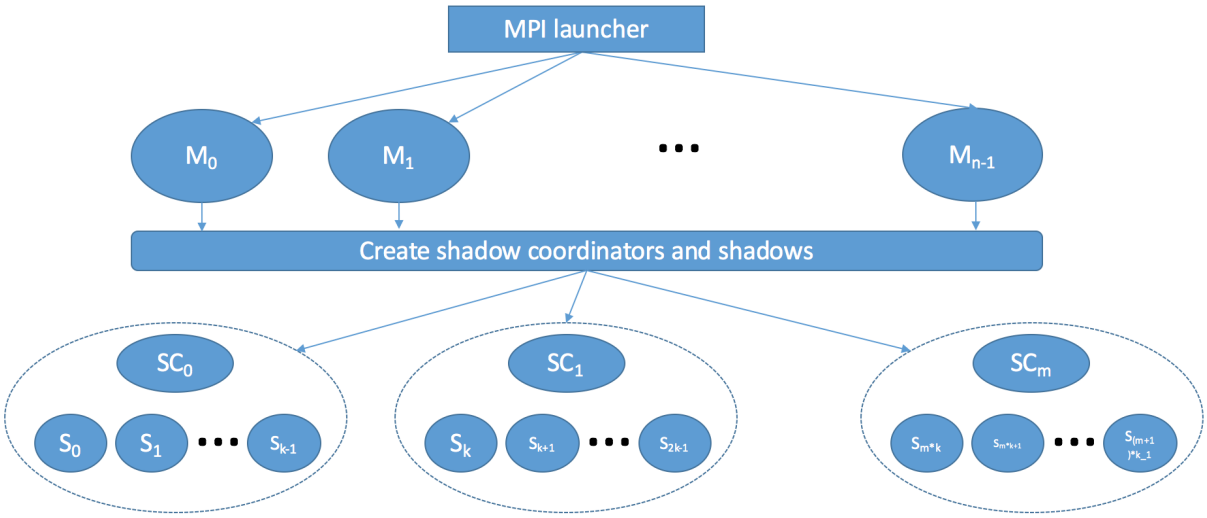
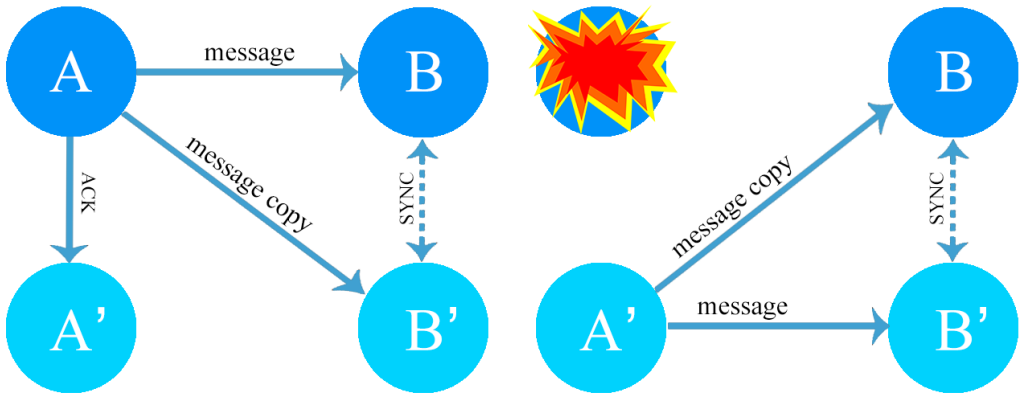


Figure 8: Logical organization of a MPI world with Lazy Shadowing.

processes of the failure.

When the user specifies N processes, lsMPI will translate it into $2N + K$ processes where K is the number of shadowed sets. The user has the flexibility to specify how the main processes are organized into shadowed sets through a rankfile. During initialization lsMPI will spawn N main processes, N shadow processes, and 1 shadow coordinator per shadowed set. The logical organization is depicted in Figure 8. Shadow coordinator manages the coordination between main and shadow processes. When a main process finishes, it will notify its corresponding shadow coordinator, which then terminates the associated shadow process. When a main process fails, the RAS system will notify the corresponding shadow coordinator, which then promotes the associated shadow process to a new main process and kills the collocated shadows.

State consistency is required both during normal execution and following a failure. We design a consistency protocol as shown in Figure 9, to assure that the shadows see the same message order and MPI operation results as the mains. In this figure, A and B represent two mains, and A' and B' are their shadows. For each message, the main of the sender sends a copy of the message to each of the main and shadow of the receiver, and the shadow of the



(a) Before failure (b) After failure

Figure 9: Consistency protocol for lsMPI.

sender is suppressed from sending out messages until its main process fails. If a main fails, its associated shadow will become a new main and starts sending out messages. To assure consistent transition, i.e., there is neither duplicate or missing message, we require the main to send an ACK after each application message.

We assume that only MPI operations can introduce non-determinism. MPI_ANY_SOURCE receives may result in different message orders between the main and shadow. To deal with this, we always let the main receive a message ahead of the shadow and then forward the message source to its shadow (SYNC message in Figure 9). The shadow then issues a receive with the specific source. Other operations, such as MPI_Wtime() and MPI_Probe(), can be dealt with by always forwarding the result from the main to the shadow.

Remote Direct Memory Access (RDMA) will be used to leap forward the state of the shadow to be consistent with that of its associated main. Rather than copying data to the buffers of the OS, RDMA allows to transfer data directly from the main process to its shadow. The zero-copy feature of RDMA considerably reduces latency, thereby enabling fast transfer of data between the main and its shadow.

After the implementation is complete, I will firstly verify its correctness by running benchmarks like NAS and Mantevo benchmark suites. Then using a combination of benchmarks

and real applications, I will measure the runtime overhead by comparing the execution time of lsMPI with that of no shadows. Lastly, I will compare lsMPI with process replication and checkpointing by running benchmarks and real applications with emulated failures.

BIBLIOGRAPHY

[1] S. Agarwal, R. Garg, M. S. Gupta, and J. E. Moreira. Adaptive incremental checkpointing for massively parallel systems. In *ICS 04*, St. Malo, France.

[2] S. Ahern, A. Shoshani, K.-L. Ma, A. Choudhary, T. Critchlow, S. Klasky, V. Pascucci, J. Ahrens, E. W. Bethel, H. Childs, J. Huang, K. Joy, Q. Koziol, G. Lofstead, J. S. Meredith, K. Moreland, G. Ostrouchov, M. Papka, V. Vishwanath, M. Wolf, N. Wright, and K. Wu. *Scientific Discovery at the Exascale, a Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization*. 2011.

[3] L. Alvisi, E. Elnozahy, S. Rao, S. A. Husain, and A. de Mel. An analysis of communication induced checkpointing. In *Fault-Tolerant Computing*, 1999.

[4] S. Ashby, B. Pete, C. Jackie, and C. Phil. The opportunities and challenges of exascale computing, 2010.

[5] J. F. Bartlett. A nonstop kernel. In *Proceedings of the Eighth ACM Symposium on Operating Systems Principles*, SOSP '81, pages 22–29, New York, NY, USA, 1981. ACM.

[6] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, K. Yelick, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Keckler, D. Klein, P. Kogge, R. S. Williams, and K. Yelick. Exascale computing study: Technology challenges in achieving exascale systems peter kogge, editor & study lead, 2008.

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

[8] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir. Toward exascale resilience. *Int. J. High Perform. Comput. Appl.*, 23(4):374–388, Nov. 2009.

[9] V. Chandra and R. Aitken. Impact of technology and voltage scaling on the soft error susceptibility in nanoscale CMOS. In *Defect and Fault Tolerance of VLSI Systems*, 2008.

[10] K. Chandy and C. Ramamoorthy. Rollback and recovery strategies for computer programs. *Computers, IEEE Transactions on*, C-21(6):546–556, June 1972.

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

[12] H. chang Nam, J. Kim, S. Lee, and S. Lee. Probabilistic checkpointing. In *In Proceedings of Intl. Symposium on Fault-Tolerant Computing*, pages 153–160, 1997.

[13] F. Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78, Feb. 1991.

[14] X. Cui, B. Mills, T. Znati, and R. Melhem. Shadow replication: An energy-aware, fault-tolerant computational model for green cloud computing. *Energies*, 7(8):5151–5176, 2014.

[15] X. Cui, B. Mills, T. Znati, and R. Melhem. Shadows on the cloud: An energy-aware, profit maximizing resilience framework for cloud computing. In *CLOSER*, April 2014.

[16] X. Cui, T. Znati, and R. Melhem. Adaptive and power-aware resilience for extreme-scale computing. In *16th IEEE International Conference on Scalable Computing and Communications*, July 18-21 2016.

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

[18] G. Deconinck, J. Vounckx, R. Lauwereins, and J. A. Peperstraete. Survey of backward error recovery techniques for multicomputers based on checkpointing and rollback. *International Journal of Modeling and Simulation*, 18:262–265, 1993.

[19] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer. Lessons learned from the analysis of system failures at petascale: The case of blue waters. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 610–621. IEEE, 2014.

[20] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, 65(3):1302–1326, 2013.

[21] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining partial redundancy and checkpointing for HPC. In *ICDCS '12*, Washington, DC, US.

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

[23] E. Elnozahy and W. Zwaenepoel. Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit. *TC*, 41:526–531, 1992.

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

[25] C. Engelmann and S. Böhm. Redundant execution of hpc applications with mr-mpi. In *PDCN*, pages 15–17, 2011.

[26] S. Eyerman and L. Eeckhout. Fine-grained dvfs using on-chip regulators. *ACM Trans. Archit. Code Optim.*, 8(1):1:1–1:24, Feb. 2011.

[27] 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, 2011.

[28] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. SC, Los Alamitos, CA, USA, 2012.

[29] F. C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26, Mar. 1999.

[30] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello. Uncoordinated checkpointing without domino effect for send-deterministic mpi applications. In *IPDPS*, pages 989–1000, May 2011.

[31] J.-M. Helary, A. Mostefaoui, R. H. Netzer, and M. Raynal. Preventing useless checkpoints in distributed computations. In *RDS*, 1997.

[32] S. Kalaiselvi and V. Rajaraman. A survey of checkpointing algorithms for parallel and distributed computers. *Sadhana*, 25(5):489–510, 2000.

[33] B. Keller. Opportunities for fine-grained adaptive voltage scaling to improve system-level energy efficiency. Master's thesis, EECS Department, University of California, Berkeley, Dec 2015.

[34] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 123–134, Feb 2008.

[35] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[36] K. Li, J. F. Naughton, and J. S. Plank. Low-latency, concurrent checkpointing for parallel programs. *IEEE Trans. Parallel Distrib. Syst.*, 5(8):874–879, Aug. 1994.

[37] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra, A. Geist, G. Grider, R. Haring, J. Hittinger, A. Hoisie, D. Klein, P. Kogge, R. Lethin, V. Sarkar, R. Schreiber, J. Shalf, T. Sterling, and R. Stevens. *The Top Ten Exascale System Research Challenges*. 2014.

[38] E. Meneses, O. Sarood, and L. V. Kalé. Energy profile of rollback-recovery strategies in high performance computing. *Parallel Computing*, 40(9):536–547, 2014.

[39] B. Mills. *Power-Aware Resilience for Exascale Computing*. PhD thesis, University of Pittsburgh, 2014.

[40] B. Mills, T. Znati, and R. Melhem. Shadow computing: An energy-aware fault tolerant computing model. In *ICNC*, 2014.

[41] B. Mills, T. Znati, R. Melhem, K. B. Ferreira, and R. E. Grant. Energy consumption of resilience mechanisms in large scale systems. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 528–535. IEEE, 2014.

[42] B. Mills, T. Znati, R. Melhem, R. E. Grant, and K. B. Ferreira. Energy consumption of resilience mechanisms in large scale systems. In *Parallel, Distributed and Network-Based Processing (PDP), 22st Euromicro International Conference*, Feb 2014.

[43] A. Moody, G. Bronevetsky, K. Mohror, and B. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC*, pages 1–11, 2010.

[44] X. Ni, E. Meneses, N. Jain, and L. V. Kalé. Acr: Automatic checkpoint/restart for soft and hard error protection. SC, pages 7:1–7:12, New York, NY, USA, 2013. ACM.

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

[46] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, Mar. 2014.

[47] J. Plank and K. Li. Faster checkpointing with n+1 parity. In *Fault-Tolerant Computing*, pages 288–297, June 1994.

[48] B. Randell. System structure for software fault tolerance. In *Proceedings of the international conference on Reliable software*, New York, NY, USA, 1975. ACM.

[49] S. K. Reinhardt and S. S. Mukherjee. *Transient fault detection via simultaneous multithreading*, volume 28. ACM, 2000.

[50] 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.

[51] A. Sangroya, D. Serrano, and S. Bouchenak. Benchmarking dependability of mapreduce systems. In *Reliable Distributed Systems (SRDS), IEEE 31st Symp. on*, pages 21–30, 2012.

[52] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, Dec. 1990.

[53] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, et al. Addressing failures in exascale computing. *International Journal of High Performance Computing Applications*, page 1094342014522573, 2014.

[54] P. Sousa, N. Neves, and P. Verissimo. Resilient state machine replication. In *Dependable Computing. Proc.* 11^{th} *Pacific Rim Int. Symp. on*, pages 305–309, 2005.

[55] J. Stearley, K. Ferreira, D. Robinson, J. Laros, K. Pedretti, D. Arnold, P. Bridges, and R. Riesen. Does partial replication pay off? In *DSN-W*, pages 1–6, June 2012.

[56] A. Tchana, L. Broto, and D. Hagimont. Approaches to cloud computing fault tolerance. In *Comp., Infor. & Tele. Sys., Int. Conf. on*, pages 1–6, 2012.

[57] W.-T. Tsai, P. Zhong, J. Elston, X. Bai, and Y. Chen. Service replication strategies with mapreduce in clouds. In *Autonomous Decentralized Systems,* 10^{th} *Int. Symp. on*, pages 381–388, 2011.

[58] B. Zhao, H. Aydin, and D. Zhu. Reliability-aware dynamic voltage scaling for energy-constrained real-time embedded systems. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, pages 633–639. IEEE, 2008.

[59] G. Zheng, X. Ni, and L. V. Kal. A scalable double in-memory checkpoint and restart scheme towards exascale. In *DSN-W*, pages 1–6, June 2012.

[60] G. Zheng, L. Shi, and L. V. Kalé. FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In *Cluster Computing*, pages 93–103, 2004.

[61] Q. Zheng. Improving mapreduce fault tolerance in the cloud. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–6, April 2010.