1 Project Description

1.1 Background and Motivation

Our Main Goal: We propose Shadow Computing as a framework to achieve the three goals of resilience to faults, energy efficiency, and Quality of Service (QoS) in likely future computing environments consisting of a large number of processors. The main goal of the proposed research is to investigate the efficacy of Shadow Computing in achieving these goals. Shadow Computing seeks to achieve resilience to faults by using redundant secondary processes, called *shadows*. Each main process is associated with a shadow that can take over execution, when the main process experiences a fault. Shadow Computing seeks to achieve energy efficiency by running shadows at a reduced rate, potentially reducing energy consumption. Shadow Computing seeks to achieve QoS by running shadows at a high enough rate that a reasonable number of faults will not prevent the completion of the task in a reasonable amount of time. As these three goals are at least partially conflicting, and the relationship between them seems delicate, achieving the full promise of Shadow Computing requires significant research and development.

The importance of achieving the goals of resilience to faults, energy efficiency, and QoS: Computing and information systems have become integral to all aspects of our life, and their significance will inevitably continue to increase in the future. Fueled by powerful computing servers, massive storage capacity and unprecedented communication speeds, these systems support an increasingly large number of services and applications, critical to our society, national security and the economy. These applications range from Internet-scale web services, high-performance scientific computing, data-intensive analytics and graph processing, mobile computing for smart distributed environments, healthcare and disaster management. As our reliance on information technology continues to grow, future infrastructure must support hundreds of thousands, if not millions, of servers, to achieve significantly higher levels of parallelism and handle a massive numbers of storage and communications operations.

The upward trend, in terms of scale and distribution, has a direct negative effect on the overall system reliability. The increase in the number of system components significantly increases the propensity of the system to faults, while driving power consumption to unprecedented heights. 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 devices, possibly by several orders of magnitude. For example, a computing infrastructure with 200,000 nodes will experience a *mean time between failure* (MTBF) of less than one hour, even when the MTBF of an individual node is as large as 5 years[82, 43, 90, 97]. With the explosive growth in system components also comes a sharp increase in system power requirements, making energy consumption a leading design constraint of future extreme scale computing systems. Recent projections show a steady rise in system power consumption to 1-3MW in 2008, followed by a sharp increase to 10-20MW in subsequent years, with the expectation that power consumption could surpass 50MW by 2016 [8]). The Department of Energy has recognized this trend and established a power limit of 20 megawatt [8], challenging the research community to provide a 1000x improvement in performance with only a 10x increase in power. It is reported that energy cost in data centers accounts for 23-50% of the overall expenses, making system power a leading design constraint in future extreme-scale computing infrastructure.

The scale needed to address our future computing needs will come at the cost of increasingly sophisticated, complex computing and information systems whose behavior is increasingly difficult to specify, predict and manage. Fault tolerance and power management have been studied extensively, although only recently have researchers begun to study the combination of these two competing goals. In today's systems the response to faults mainly consists of restarting the application, including all related software components that have been affected by the fault. To avoid full re-execution of the original task, systems often checkpoint the execution periodically. Upon the occurrence of a hardware or software failure, recovery is then achieved by restarting the computation from a known checkpoint. Given the anticipated increase in system-level failure rates and the time required to checkpoint large-scale compute-intensive and data-intensive applications, it is predicted that, in extreme scale computing environments, 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 performance of the system[82, 43, 90]. Given this limitation, several large-scale application designers avoid checkpointing and rely on re-execution, upon failure, to achieve fault-tolerance. More recently process replication, either fully or partially, has been proposed as an

alternative to checkpointing and rollback recovery in HPC environments [44, 97, 20, 23, 70, 40, 22, 43]. It has also been used in Cloud Computing systems to tolerate server as well as interconnection failures[12, 91, 59, 110]

The difficulty of achieving resilience, energy efficiency, and QoS: There is a delicate interplay between fault-tolerance and energy consumption. Checkpointing, re-execution and replication require additional energy to achieve fault-tolerance. Conversely, it has been shown that lowering supply voltages, a commonly used technique to conserve energy, increases the probability of transient faults [24, 54, 112]. The tradeoffs between fault free operation and optimal energy 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 energy consumption dichotomy. For example, it has been shown that, at scale, coordinated and uncoordinated checkpointing may lead to cascading delays. Similarly, the overhead of reexcution and replication-based techniques is a dominant factor of the computation and may lead to excessive delay and energy consumption in large-scale computing environments. Abrupt and unpredictable changes in system behavior may lead to unexpected fluctuations in performance, which can be detrimental to applications' QoS requirements. The high risk of relying on computing systems that are fragile and cannot deliver the QoS performance for which they were engineered, calls for new resilience frameworks to ensure a high-level of availability and dependability. We take the view that failures in emerging large-scale systems will be the norm, rather than the exception, and seek to develop uniform adaptive frameworks, methodologies and tools to achieve QoS-resiliency in future computing systems. In this project, QoS-resiliency refers to the ability of the system to achieve a specified level of fault-tolerance and meet the task completion time, while minimize energy consumption.

How Shadow Computing conserves energy: As a simple example of Shadow Computing, and its potential for energy savings, consider the following scenario. The setting is two processors whose speed may be scaled between 0 and 2. We assume we wish to complete one task with work w by some deadline t_s. The dynamic power used when running at speed s is s^3. We seek algorithms that are tolerant of at least one fault. Assume that faults occur with a probability p and a fault is equally likely to happen on either processor at any time. With probability $(1-p)$ there is no fault.

We consider the following class of algorithms. The algorithm runs the main copy of the task on one processor at speed σ_m and runs a shadow copy on the other processor at some speed $\sigma_s \leq \sigma_m$; If the main copy experiences a fault, then the shadow becomes the main and is **sped up if necessary** so that the task will complete at time t_s. When the main task completes, then all speeds are set to 0. One gets different algorithms depending upon the speeds σ_m and σ_s, but all algorithms will always finish the job if there is at most one fault. Without loss of generality, after rescaling w and t_s each to 1, one can see by straight-forward calculations that the expected energy of such an algorithm is:

$$(p/2)\left[\int_0^{1/\sigma_m}(\sigma_m^3+\sigma_s^3)t_f+(1-t_f)\left(\frac{1-t_f\sigma_m}{1-t_f}\right)^3 dt_f+\int_0^{1/\sigma_m}\sigma_s^3\cdot t\ dt_f\right] \quad + \quad \left(1-\frac{1}{\sigma_m}\right)\left(\frac{1}{\sigma_m}\right)(\sigma_m^3 \quad + \quad \sigma_s^3)$$

Using Mathematica, one can see the advantage of shadow computing, namely that the algorithm that optimizes expected energy sets σ_s to some speed strictly intermediate between 0 and σ_m. For example if $p = 1/2$, then the optimal strategy is when $\sigma_m \approx 1.04$ and $\sigma_s \approx 0.79$. The execution dynamics of shadowing are depicted in Figure 1.

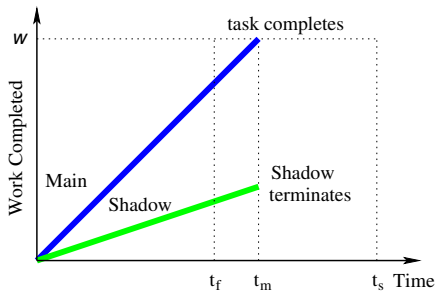
More generally, the execution rates of the shadow can be derived by balancing the trade-offs between completion time and energy consumption. For a delay-tolerant, energy-stringent application, the optimal shadow rate would be 0, and the shadow starts executing only upon failure of the main process. For a delay-stringent, energy-tolerant application, the optimal shadow rate would be the same as the main to guarantee the completion of the task by its deadline.

Shadow Computing provides a framework to balance resource redundancy, both in time and hardware, in order to meet the application's expected levels of QoS performance and reduce energy consumption, while operating under the specified power constraints of the computation environment. To understand and realize the full potential of Shadow Computing in future large scale computing environments, we propose the following three pronged research agenda.

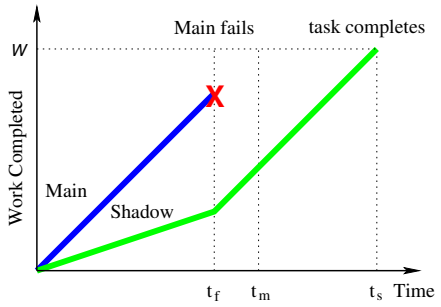
In the first thrust, we will develop models that are simple enough for theoretical study. Our initial models will assume that faults are stochastic with known distributions. It is likely that, as in most stochastic optimization problems, the optimal strategies in these models are too complicated to compute, to understand, or even to describe in polynomial space. Thus we propose to seek stochastically competitive algorithms, which guarantee small relative error with

respect this the optimal in terms of the expected cost. The expectation is that the goal of stochastic competitiveness will guide the algorithm development, and that the derived algorithms will be the basis of the algorithms that we will simulate and implement. We also expect to prove bounds that set the limits for what might be achievable in practice.

In the second thrust, guided by the findings of the theoretical study we will extend the basic Shadow Computing to support resilience in more realistic parallel-execution environments, taking into consideration the coordinated execution of tasks and the impact of communication and synchronization, using different reliability models and application-dependent objective functions. We will investigate adaptive approaches and develop efficient runtime mechanisms, algorithms and data structures to to achieve energy-efficient state consistency between the main and the shadow processes. In the third thrust, we will verify, test and evaluate the techniques developed in the second thrust using simulation and a prototype implementation in high-performance and cloud computing environments. The specific tasks associated with each thrust are described in the management plan section of the proposal.



(a) Main process successful completion



(b) Main process failure

Figure 1: Shadow Computing execution dynamics.

1.2 Underlying Theory of Shadow Computing

The high level goal of this theoretical investigation is to understand to what extent the demands of fault tolerance, energy efficiency and quality of service can be balanced. PI Pruhs will lead this investigation. He has published extensively on theoretical problems related to scheduling to balance energy efficiency and quality of service, some examples since 2012 include [57, 14, 13, 66, 16, 29, 15, 30, 17, 48, 51]. He also has published on theoretical problems related to scheduling to balance fault-tolerance and quality of service [63, 62]. He also participated in several system-oriented research projects.

There are two standard high level approaches for analyzing algorithms in settings such as these. The first standard approach is a worst-case adversarial approach, commonly called competitive analysis. In such an approach no probabilistic assumptions are made about what events. The goal is then to find algorithms that guarantee a bounded relative error in comparison to the optimal objective value that could have been achieved given full knowledge of the future. This relative error is commonly called the competitive ratio. The use of competitive analysis for analyzing algorithms within the context of shadow computing seems problematic as the worst-case scenario will generally be that a fault is experienced right before an algorithm finishes a job. Thus there is a danger that competitive analysis will suggest unrealistic algorithms that focus too much on this worst-case fault scenario. The second standard approach, is to assume that the events seen by the algorithm are drawn from some mathematically simple distribution. Let us call this the stochastic optimization approach. Within the context of stochastic optimization, one might then want to analyze the expected cost of natural algorithms, and/or seek algorithms that optimize the expected cost. Unfortunately the norm in stochastic optimization problems is that the optimal algorithm is too complicated to compute, or even understand. Often there is not even a polynomial size description of the optimal algorithm. One way to deal with this, which we propose to adopt, is to seek algorithms that guarantee small relative error with respect to the optimal in terms of the expected cost. Let us call such algorithms stochastically competitive.

For example, this is the approach taken to analyze stochastic knapsack (and related) problems [33, 49, 79, 47, 50, 18, 19, 71]. Consider the stochastic knapsack problem where the algorithm must sequentially pick items with known rewards and stochastic sizes (the distributions are known a priori but the instantiated size is only known after selection). The knapsack has a capacity that specifies the aggregate size limit of items that can be included. When the aggregate size of the selected items exceeds the capacity, this last item, and all subsequent items are discarded. The objective is to maximize the expected reward. In stochastic optimization problems, such as stochastic knapsack, one

commonly seeks not only stochastically competitive algorithms, but also algorithms that minimally adapt to stochastic events. Presumably such algorithms will be simpler and incur less overhead when implemented. Such algorithms are usually called *non-adaptive*, where the definition of non-adaptive depends upon the problem at hand. In the context of stochastic knapsack, the natural definition of non-adaptive is that the order that the items is considered is not affected by stochastic events. The authors of [33] gave randomized nonadaptive stochastically competitive algorithm for stochastic knapsack. With some probability this algorithm picks the most valuable item first, and otherwise it picks the items by decreasing ratio of reward to expected size.

1.2.1 Research: Some general questions

We propose to theoretically investigate balancing energy efficiency, fault tolerance, and quality of service by seeking stochastically competitive algorithms for a sequence of increasing complex problems. Before describing these concrete problems, let us discuss some general theoretical questions that we hope will be answers by working on these problems.

Understanding the effect of the fault rate/probability: In the simple example instance from the introduction, the energy optimal way to achieve fault tolerance not surprisingly depends on the probability p of a fault. If $p = 0$, then no shadow was needed, and if $p = 1$, the optimal solution was to run the shadow at the same rate as the primary. One interesting question is to understand how the optimal schedule depends upon the fault rate/probability. For example, does the optimal schedule evolve in some sense continuously as a function of p? In the context of balancing energy efficiency and quality of service, it is know that for some quality of service objectives the optimal schedule evolves continuously with respect the relative value of energy and quality of service, and for other quality of service objectives the optimal schedule does not evolve continuously [29, 13]. It would also be desirable to find algorithms that do not need to know exactly the fault rate/probability. Thus it would be interesting to determine how accurate of a estimate of the fault rate/probability is need to achieve stochastic competitiveness. In particular, is it possible to achieve stochastic competitiveness without a priori knowledge of the fault rate/probability?

Understanding the effect of the speed-power relationship: If $P(s)$ is the power that must be used to obtain speed s, then $P(s)/s$ represents the energy efficiency as it gives the amount of energy used per unit of work when run at speed s. In the unrealistic scenario that $P(s)$ is linear, and thus $P(s)/s$ is constant. Thus no energy efficiency is gained by running shadow copies. But typically $P(s)$ is strictly convex. As we have seen, when $P(s) = s^3$, one can provide more energy efficient fault tolerance using a shadow. It would be interesting to understand the relationship between the convexity of $P(s)$ and how to achieve energy-efficient fault tolerance. If there are no faults, it is known that when scheduling jobs with deadline feasibility constraints, the optimal competitive ratio must be a function of the convexity of the power function [15]. It is also known that when the objective is a linear combination of energy and response time that the convexity of the power function does not really affect the achievable competitiveness [16]. In the context of shadow computing, it is not clear whether the competitiveness must necessarily increase with the convexity of the power function.

Minimizing Adaptivity: It would be interesting to determine how much adaptivity is required in order for an algorithm to be stochastically competitive . There does not seem to be a unique natural definition of non-adaptive in the setting of shadow computing. One natural candidate for definition of non-adaptive is that speeds only change/increase in response to a fault, and the new speed only depends on the time of the fault, and the state of other copies of this job. Another natural, more minimal, candidate definition for non-adaptive is that a copy of a job will not change speed until some copy of that job is finished. The hope is that by looking at several concrete problems we will obtain some understanding about which types of adaptivity are helpful and which ones are not so helpful to an algorithm.

Algorithmic Dependence on Other Jobs: Presumably we will be able to develop some reasonable understanding of a good algorithm to achieve fault-tolerance for one job in an energy efficient manner. However, it is not at all clear how the existence of other jobs will effect this algorithm. For example, will the existence of other jobs decrease the amount of energy that one should devote to the reliability of an individual job? If so, how does the rate of this decrease scale with the number of other jobs in the system?

The Optimal Adaptive Algorithm: The main mathematical difficulty in showing that algorithms are stochastically competitive is getting some handle on the optimal adaptive algorithm. The standard approach is to find some simple, succinct lower bound. For stochastic knapsack problems, the standard lower bounds are linear programs. A natural

research question in the context of shadow computing is going to be what to use for lower bounds for the optimal adaptive algorithm. In particular, to determine whether some linear program can be used. The fact that linear programming formulations exist for some of the problems that we consider if there are no faults gives some hope that this may be possible.

1.2.2 Research: Some Specific Questions

We consider a sequence of seemingly increasing complexity from the simple example problem in the introduction.

Generalizing to multiple processors: This is the generalization of the simple example problem in the introduction to the case that there are many processors. We will need to change the fault model so that the expected number of faults scales with the number of processors. One possibility is to have a parameter k specify the number of faults that must be tolerated, and assume that each machine is equally likely to fault, and conditioned on a processor faulting, each time for that fault is equally likely. Another possibility is to assume that the time to the first fault on each machine is independently and exponentially distributed with mean λ. The first research goal would be to determine if the optimal adaptive algorithm is simple enough to be determined. If not, we would seek a minimally adaptive stochastically competitive algorithm.

Generalizing to multiple jobs: Once we have built some intuition for how to achieve fault-tolerance in an energy efficient manner for one job, we will need to generalize to handling many jobs. Initially we will assume that each job j has a release time r_j when it arrives in the systems, a work/size w_j and a deadline d_j by which it must be completed. It seems more natural in this setting to consider transient faults. So a job may be restarted on a processor immediately after it faults. It seems most natural to assume that the distribution on faults on each processor is Poisson. Again we seek an algorithm that completes each job by its deadline, and that the objected is expected energy usage. Again we would seek minimally adaptive stochastically competitive strategies. A useful starting point presumably would be the competitive strategies in the case of no faults in [9].

Generalizing to inter-processor power heterogeneity instead of intra-processor power heterogeneity: As the number of processors per chip increases over time, it seems increasingly likely that intra-processor power heterogeneity (speed scaling) will be replaced by inter-processor power heterogeneity. In particular, it is likely a chip will contain a small number of high-speed energy-inefficient processors, a large number low-speed energy-efficient processors, and maybe an intermediate number of processors with intermediate speed and energy efficiency. A reasonable initial model is to assume that the ith processor runs at speed speed s_i using power P_i. Presumably we would also want to allow the possibility that the reliability of the processors is also heterogeneous. So in the case of one job, we could assume that faults on each processor i has an exponentially distributed time to failure with some mean λ_i. And in the case of multiple jobs, we could assume that the faults on each processor i have a Poisson distribution with parameter λ_i. Generally speaking inter-processor power heterogeneity is much harder to handle theoretically than intra-processor power heterogeneity. The main reason is that the number of jobs that can be run at high speed is fixed, and thus it is important to efficiently unitize the low-speed processors. As now processors speed have upper limits, one can not longer guarantee that a job will be finished. This could be addressed in several ways. One option would be to allow a job i to miss its deadline, but this would incur some penalty p_i. Then one might consider the objective of expected energy plus aggregate penalty. Again the objective is to find stochastically competitive algorithms, and presumably a good starting point is the algorithms that are known to be competitive without faults [64].

Generalizing from deadline feasibility constraints to total/average response time: The initial theoretical research into scheduling to balance energy efficiency and quality of service focused on the quality of service objective being formalized as deadline feasibility constraints because generally this is the easiest of the standard quality of service formulations to handle mathematically; But eventually a theoretical understanding of the more common quality of service objective of average/total response time was developed. When the quality of service is an objective (not constraints) most theoretical research assumes that there exist a parameter β, specified by either the user or the application, that specifies the relative importance of energy vs. quality of service. More formally, β specifies the amount of energy that the system can/should invest to reduce the quality of service objective by a unit amount. The resulting objective is then to minimize some quality of service objective plus β times the energy used by the processor. The optimal schedule for this objective is then the optimal energy trade-off schedule in the sense that: No schedule can have better quality

of service given the current investment of energy, and an additional investment of 1 unit of energy is insufficient to improve the quality of service by more than β. Again the goal would be to find stochastically competitive algorithms, and presumably a good starting point would be the natural competitive algorithm when there are no faults, which maintains the invariant that the power is proportional to the number of jobs in the system [16]. In some sense [16] shows that the optimal policy when all jobs arrive at the same time is also optimal if jobs arrive over time. It would be interesting to determine if this also holds in the case of faults.

1.3 Enabling Shadow Computing at Scale

A number of programming models have been proposed to enable the design and efficient implementation of distributed and parallel computing over a large-scale infrastructure[34, 45, 80, 58]. Rather than focusing on a specific programming model, such as Message Passing Interface (MPI) or Mapreduce, we adopt the Bulk Synchronous Parallel (BSP) computational model, as the building block for the class of applications considered in this research [105]. A BSP computation consists of a sequence of *supersteps*. Each superstep involves three main activities: **computation**, **communication** and **synchronization**. At the end of the synchronization interval, a global check is performed to ensure that all supersets have been completed by all the components, before proceeding with the next superstep, until completion.

We consider a class of BPS-inspired applications, executing on a large-scale computing infrastructure [8]. Communication between computing nodes is achieved using low-latency, high-bandwidth interconnect networks. We use W to denote the size of an application workload, and assume that the workload is split arbitrarily into a set of components; each component, i, is associated with a task, $T(i)$. All tasks execute in parallel and use barriers for synchronization. Communication between tasks is based on message passing, whereby each pair of communicating tasks is associated with a logical FIFO channel, to guarantee ordered delivery of messages.

Shadow Computing provides the basis for the design of resilience models, which take into consideration energy, reliability and delay tradeoffs. The basic model, however, focuses on a single task, and does not address the communication and synhcronization of group of tasks cooperatively executing in parallel to complete a job. To realize the vision of Shadow Computing in large-scale, multi-task environents, there are a number of challenges, beyond those raised in the theoretical study of this project, that we must address:

- Extend the basic Shadow Computing model to address the communication and synchronization requirements of BSP-inspired computation, operating in failure-prone environments. The objective is to mitigate the impact of failure on the performance of non-failing processes and ensure forward progress of the computation.
- Develop mechanisms to determine and control the shadow's excurion rates, both before and after the failure of the main process. These mechanisms must account for the capabilities of the computing infrastructure.
- Develop mechanisms and data structure to efficiently support the interaction between the main and its shadow. Message flow control and state consistency must be synergistically cooridinated.
- Investigate adpative runtime optimization techniques to reduce communication and improve performance.

1.3.1 Extending Shadow Computing for BSP computation

To highlight the impact of the combined effect induced by synchronization and failure on BSP computation, consider a job with N tasks, $M(i)(1 \leq i \leq N)$ and their N shadows, $S(i)(1 \leq i \leq N)$. Figure 2 shows the behavior of two main processes and their associated shadows, upon the occurrence of a failure at time t_f. Figure 2(a) depicts the execution dynamics of a failing main process, $M(i)$, and its shadow, $S(i)$, while Figure 2(b) illustrates the behavior of a non-failing main process, $M(j)$ and its shadow, $S(j)$. Upon $M(i)$'s failure, $S(i)$ continues execution, but at a higher rate, until it reaches the synchronization barrier at time t_r. Unaware of $M(i)$'s failure, $M(j)$ continues to execute until it reaches the synchronization barrier, at time t_{sync}. Note, however, that while $S(i)$ progresses toward the synchronization barrier, $M(j)$ remains idle until time t_r.

The combined effect of synchronization and failure on non-faulty processes waiting for the shadow of a faulty main becomes more acute in environments where synchronization occurs frequently. To address this shortcoming, we propose an enhanced instance of the Shadow Computing model, referred to as *Leaping Shadows*. In this model, shadows opportunistically take advantage of the failure-induced idle time to *leap forward* and consolidate their execution state with that of their associated main. After the shadow of the failing main reaches its synchronization barrier, all processes resume execution from a consistent synchronization point. This process continues until the completion of all tasks. Forward 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 its associated main, thereby ensuring fast recovery, upon failure, while minimizing energy consumption.

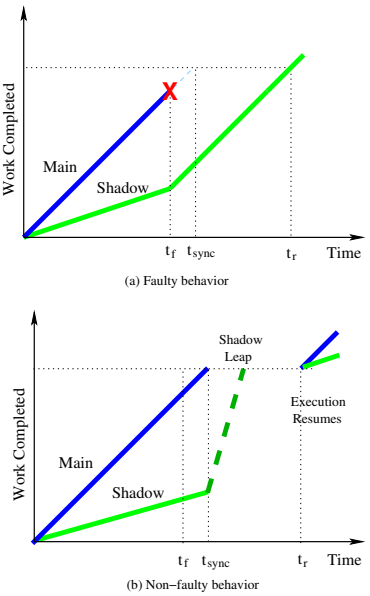


Figure 2: Execution dynamics under failure

1.3.2 Shadow execution rates

Three execution rates must be determined in Shadow Computing: the main process execution rate, σ_m, and the before and after failure execution rates, $\sigma_s^b \leq \sigma_m$ and σ_s^a, of its associated shadow. In the absence of failure, the main process completes execution at time $t_m = w/\sigma_m$, where w represents the size of the task's workload. However, if at time $t_f < t_m$ the main process fails, the shadow, which has completed an amount of work $w_b = \sigma_s^b * t_f$, increases its execution rate to σ_s^a to complete the task by t_s. The values of σ_s^b and σ_s^a can be derived by balancing the trade-offs between completion time and energy consumption. The derived values are then used to set a shadow's execution rate, so that the task completes in time, even in the presence of failure. In the following, we describe our approach to determine and control the shadows' execution rates.

Optimal execution rates: The main objective is to compute the optimal execution rates of the main and its shadow in order to meet application QoS-resilience, while minimizing energy, possibly under power-constraints. In its most generic form, this optimization problem can be formulated using the following basic structure:

$$\text{Maximize} \quad \sum_{p \in \mathbf{J}} \mathbf{U}(p, \sigma_m^p, \sigma_b^p, \sigma_a^p)$$

$$\text{Subject to} \quad \sigma_m^p T_c^p \geq W^p \qquad \text{Main process work conservation constraint}$$

$$\int_0^{T_c^p} \left(\sigma_b^p t + \sigma_a^p (t_s^p - t)\right) \Phi(t) \mathrm{d}t \geq W^p \qquad \text{Shadow process work conservation constraint}$$

$$\sigma_m^p, \sigma_b^p, \text{and } \sigma_a^p \in \mathcal{S} = \{\sigma_i,\ 1 \leq i \leq l\} \qquad \text{Operating execution rates constraint}$$

In above formalization, $\mathbf{U}()$ is the utility derived from the completion job $\mathbf{J}$, consisting of multiple tasks, $p \in \mathbf{J}$. T_c^p is the task main process completion time, W^p the workload of process p, t_s^p the shadow's completion time, l the operating levels of discrete frequencies, and $\Phi()$ is the failure distribution. The type and nature of the utility functions depends on the optimization objective. Different types of utility fuctions will be considered, including energy cost, penalty for delayed completion and service level QoS requirements. The output of this optimization problem is the execution rates, σ_m^p, σ_b^p and σ_a^p, of the main and its shadow for task, p. Clearly, the above optimization problem is non-linear and discrete in terms of the decision variables for both the objective and the constraints of the problem. We will build on the outcome of the first thrust of this project to find efficient heuristics to solve these optimization problems. We will also validate our model, using extensive simulation studies using different failure distributions and QoS-requirments. We will investigate different application workloads, running under different load conditions, over heterogeneous, power-constraint computing infrastructure.

Execution rate control: Dynamic Voltage and Frequency Scaling (DVFS), a commonly used power management technique, can be used to control the shadow's execution rates. Assuming a computing node has the ability to operate at l discrete frequencies, $\mathcal{F} = \{f_i, 1 \leq i \leq f_l\}$, the shadow's execution rates can be dynamically scaled to the closest frequency that achieves these rates. Such a control can be completely decentralized and performed independently by each computing node. If a job is distributed to N tasks, which are executed in parallel on N cores and each of the N tasks is to be shadowed, then $2N$ cores are needed for execution: N to execute the main tasks and N to execute the shadows at the specified rates, using DVFS. Although the availability of such a technique in different computing platforms is increasing, its effectiveness may be markedly reduced in computational platforms that exhibit saturation of the processor clock frequencies, large static power consumption, or small power dynamic range. An alternative method to DVFS is *collocation*, whereby multiple processes share a core, executing at maximum rate. Such an approach to control execution rates is *natural* in virtualized environments. In this project, we will consider both alternatives for Shadow Computing. However, due to space limitation, we will focus the rest of our discussion of the main ideas, concepts and design on shadow colocating.

To highlight the basic features of collocation, assume that a job is distributed to N tasks and executed in parallel on N cores. Rather than using $2N$ cores, collocation only requires $N+K$ cores, where N is a multiple of K. The K are used to colocate N/K shadows on each core, while executing all the cores at the maximum speed. Collocating shadows effectively reduces the maximum execution rate of each shadow by a factor of K/N. Ignoring the overhead of context switching, the two alternatives lead to the same expected execution time, but result in different power and energy consumption. Specifically, the $2N$-cores alternative consumes more static but less dynamic power/energy than the N+K core alternative, since it uses more cores but applies DVFS. The ratio between static and dynamic power consumption in the system determines which alternative consumes less overall power/energy. Note that static power consumption accounts for all the non-core components of the system. We will assume that all cores in the system execute at maximum speed, with $S = N/K$ shadows executing on a single core. In terms of execution rates, this can be expressed as $\sigma_m = \sigma_a = 1$ and $\sigma_s = 1/S$. For example, if $N = 64$ and $K = 16$, then the 64 shadows execute on 16 cores, with $S = 4$ shadows executing on a single core at $1/4$ of the maximum execution rate. Collocating of S shadows on a single core has an important ramification with respect to the resilience of the system. Specifically, to execute a shadow of a failed main at maximum rate, all other colocated shadows must be terminated. Consequently,a second fault in any of the mains of the terminated shadows cannot be tolerated. In other words, the $N+K$ cores are grouped into K sets, which we call **shadowed sets**, each containing $S+1$ cores with S mains executing on S cores and their corresponding S shadows overloaded on one core. Each shadowed set can tolerate a fault in any of its cores since the role of a failing main will be assumed by its shadow and the failure of the core that executes the shadows will not affect any main. After a fault occurs in one of the cores of a shadowed set, the set is called **vulnerable** because it cannot tolerate a fault in another core. Later in this section, we will discuss a rejuvenation scheme for dealing with vulnerable shadowed sets.

M(0)	**M(2)**	**M(4)**	**M(1)**	**M(3)**	**M(5)**	**M(8)**	**M(10)**	**M(12)**	**M(9)**	**M(11)**	**M(13)**
M(6)	**S(8,9)**	**S(10,11)**	**M(7)**	**S(12,13)**	**S(14,15)**	**M(14)**	**S(0,1)**	**S(2,3)**	**M(15)**	**S(4,5)**	**S(6,7)**

(a) A mapping with shadow sets of size 3.

~~**M(0)**~~	**M(2)**	**M(4)**	**M(1)**	**M(3)**	**M(5)**	**M(8)**	**M(10)**	**M(12)**	**M(9)**	**M(11)**	**M(13)**
M(6)	**S(8,9)**	**S(10,11)**	**M(7)**	**S(12,13)**	**M(14)**	~~**M(14)**~~	**M(0)**	**S(2,3)**	**M(15)**	**S(4,5)**	**S(6,7)**

(b) If $M(0)$ and $M(14)$ fail, then shadows $S(0,1)$ and $S(14,15)$are promoted to function as mains at maximum rate.

~~**M(0)**~~	**M(2)**	~~**M(4)**~~	**M(1)**	**M(3)**	**M(5)**	**M(8)**	**M(10)**	**M(12)**	**M(9)**	**M(11)**	**M(13)**
~~**M(6)**~~	**S(8,9)**	~~**S(10,11)**~~	**M(7)**	**S(12,13)**	**S(14,15)**	**M(14)**	**M(0)**	**M(2)**	**M(15)**	**M(4)**	**M(6)**

(c) Six shadowed sets are vulnerable after a socket failure.

Figure 3: An example for mapping 16 mains and their shadows to 4 sockets of 6 cores each.

The failed component type whose failure is to be tolerated, namely a core, a socket or a node, determines the mapping of shadowed sets into the actual architecture. Specifically, the mapping should guarantee that the shadow of a failing main does not reside on the same failed component. For example, assuming that a node contains 4 sockets, each

M(0)	M(2)	M(4)
M(6)	M(8)	M(10)

M(1)	M(3)	M(5)
M(7)	M(9)	M(11)

S(0,1)	S(2,3)	S(4,5)
S(6,7)	S(8,9)	S(10,11)

Figure 4: An example of socket level shadowing.

with 6 cores, Figure 3(a) shows a mapping of 16 mains and their corresponding shadows on the 24 cores. In this mapping, the cores executing two consecutive mains, $M(i)$ and $M(i+1)$, where i is even, and the core executing their two shadows, denoted by $S(i,i+1)$, form a shadowed set $\{M(i),M(i+1),S(i,i+1)\}$. This mapping can tolerate any single core failure in a shadowed set as for example the failure of $M(0)$ and $M(14)$ shown in Figure 3(b). It can also tolerate the failure of any one of the 4 sockets, as shown in Figure 3(c). In the example of Figure 3(b), sets $\{M(0),M(1)\}$ and $\{M(14),M(15)\}$ are vulnerable after shadows $S(0,1)$ and $S(14,15)$ are promoted to function as $M(0)$ and $M(14)$, respectively, and in the example of Figure 3(c), four of the shadowed sets are vulnerable while only sets $\{M(12),M(13),S(12,13)\}$ and $\{M(14),M(15),S(14,15)\}$ are not vulnerable.

An alternative mapping that is specifically designed to deal with socket failure is to use the cores in one socket to shadow the cores in other sockets as shown in Figure 4. Recovery is more regular with this organization after a socket failure, at the price of having unbalanced memory demands among sockets. Specifically, the memory hierarchy in each of two sockets has to support six processes (mains) and support twelve processes (shadows) in the third socket. Increased memory pressure on the socket executing the twelve shadows will translate to potentially more demand paging if each socket has its own memory module, or to potentially more cache thrashing if memory is shared among the sockets of the same node. The net effect would be a degraded execution rate, σ_s, of each shadow from $1/S$ to $1/g(S)$ for some superlinear function $g(S)$. We will explore possible forms for $g(S)$ but given that cache miss rate is usually proportional to the square root of the cache size, it is reasonable to set $g(S) = S^{1.5}$, as a first approximation.

1.3.3 Dealing with communicating tasks

The communication model used in Shadow Computing assumes that the underlying hardware consists of a collection of cores connected through a high-speed network. An application using shadowing for resilience requires a runtime environment to ensure that any execution, even in the presence of failures, finishes correctly. Such an environment includes, (i) a protocol to keep a shadow's sate consistent with that of its main, (ii) a mechanism to store the necessary amount of messages, (iii) a fault detection and notification mechanism, and (iv) a recovery method that ensures the system reaches a consistent state after a failure.

It is expected that a shadow, S(i), that runs at a slow rate, lags behind its associated main, M(i). This can cause divergence between M(i)'s and S(i)'s state. Therefore, it is necessary to ensure that S(i) is capable of reaching the same state as the state of M(i) at the time the failure occurred. To this end, we will use a message-logging protocol [11] to satisfy such a requirement. Message-logging protocols use meta-information to store and replicate the non-deterministic decisions in the execution of an application. For example, if main tasks M(j) and M(k) send messages m_j and m_k to M(i), the same two messages should reach S(i). In this project, we will explore different alternatives to guarantee this property. Since message reception is, in general, non deterministic, the order in which M(i) processes m_j and m_k has to be replicated by S(i). Otherwise, S(i) may deviate from M(i)'s state. With few bits of information, message-logging protocols are able to maintain the consistency between main and shadow tasks. These pieces of information, also called determinants, are sent through system-level messages. We also must emphasize that shadows are mute, in the sense that they suppress all outgoing messages.

To provide correct recovery after failure, a mechanism is required to guarantee that the system reaches a consistent state. This requires that current message queues are flushed, missing messages are replayed and all data structures are cleared. After a main task M(i) fails, the failure notification mechanism informs all tasks of this event. S(i) then takes over M(i)'s role. Other shadows sharing the same core with S(i), if any, are terminated. S(i) can then start consuming the messages in its receiver-side message log, but at a faster rate. The message logging protocol will ensure the new main, S(i), reaches a consistent state with the rest of the system. The other shadows will roll-forward by leaping to the state of their corresponding main tasks. To achieve this, a mechanism must be developed to ensure that the state

from each main is transmitted to its shadow, message queues are flushed and other data structures are kept consistent. It is worth noting, however, that because a shadow task executes at slower rate than its main, the size of the message queues is likely to continue to grow, particularly in cases where message creation is faster than consumption. In the following, we will discuss an approach to address this problem.

1.3.4 Forced shadow leaping for efficient logging

In the absence of faults, the size of the message buffers of the slow-executing shadow continues to increase, potentially causing the overflow of the message queue. A strategy must be in place to overcome this problem. Such a strategy depends on the buffer size and the rate at which messages are consumed. To assess the impact of buffer size constraints, we executed a number of representative high-performance computing workloads and used the rMPI replication library to collect the mean message logging growth rate of these applications [42]. These workloads include the production applications CTH [35], a shock physics code, LAMMPS [95], the molecular dynamic code, the algebraic multi-grid solver AMG [68], and two of the mini-applications from Sandia's mantevo suite HPCCG, a conjugate gradient solver, and miniFE, an implicit finite element method [98]. These applications use a range of computational techniques, which are frequently run at very large scales on leadership-class systems. We measured the maximum message log growth rate for these applications, which varied from 7.43×10^5 to 5.42×10^7 Bps. The results indicate that message log growth rates can vary dramatically. For example, CTH, which does a good deal of bulk data transfer, has the largest growth rate of the applications tested, a measured 54MB/sec, while AMG, which does much less communication, has a log growth rate of nearly 1MB/sec. The message log rate increases at $1 - \sigma_s$ of the HPC mesasge rate. These results underscore the need for an efficient strategy to flow-control message-logging queue. One possible method is to set a **lower bound** on the speed of the shadows. This bound forces the shadows to execute fast enough to consume messages at a rate that prevents buffer overflow. However, such a strategy may increase unnecessarily energy consumption.

A closer look at the problem reveals that fault-induced leaping alleviates considerably the buffer overflow problem, as message queues are flushed during after states are consolidated. Therefore, what needs to be addressed is the case of a slowly-executing shadow that lags considerably behind its main, potentially resulting in message-queue overflow. To address this we propose to explore an approach, whereby lagging shadows are **forced** to leap. Forced-leaping allows lagging shadows to catch up with their mains and flush its buffers. We will develop and investigate strategies, based on buffer occupancy thresholds and message consumption rates, to trigger forced leaping. We will use mechanisms similar to those needed to achieve fault-induced leaping for the implementation of these strategies. These mechanisms are discussed in Section 1.3.6.

1.3.5 Shadowed set rejuvenation

The proposed lazy shadowing scheme can tolerate faults which are repairable by rebooting or reconfiguration, referred to as transient faults, and faults which cannot be repaired by rebooting or reconfiguration, referred to as permanent faults. Monitors that detect hardware faults, such as memory flip, bus error and latch error, or software faults, such as deadlock detection, buffer overflow and protection violation, typically interrupt the software to initiate the recovery process. The process of recovery from transient or permanent faults is the same and necessitates a mechanism for detecting a fault in a main task, $M(i)$ and notifying other tasks in the system so that (i) the shadows sharing a core with $S(i)$ are terminated, thus allowing $S(i)$ to execute at the maximum rate, and (ii) all the shadows that are not in the faulty shadowed set update their states to match the state of their mains.

As described earlier, the recovery from a fault in a shadowed set leaves the set vulnerable and any more faults in a vulnerable set will result in a system failure. Although for large systems and small S the probability of having a second fault in a vulnerable set is low, some provision should be taken to rejuvenate the system when a relatively large number of its shadowed sets are vulnerable.

We propose to invoke **Shadowed set rejuvenation** after a specific number of faults, determined by the system size, the shadowed set size and the required resilience. Rejuvenation reconfigures the system such that none of its shadowed sets are vulnerable. Unlike recovery from a fault in a shadowed set, rejuvenation is different when the faults are transient than when the faults are permanent. In the case of transient faults, rejuvenation can be accomplished by rebooting the failed cores, migrating the shadows that replaced the failed mains to their original cores and restarting

missing shadows in the shadowed sets. This will restore a vulnerable shadowed set to its original configuration. For example, rejuvenation should restore the systems shown in Figure 3(b) and Figure 3(c) to the one shown in Figure 3(a).

When failures are permanent, rejuvenation may be challenging if rebooting or reconfiguration can no longer be used to recover failed components. Specifically, in the absence of spare components (if the system is not over-provisioned), rejuvenation can only be accomplished by distributing the main processes of a vulnerable set to other **non-vulnerable** shadowed sets. The shadow of the vulnerable set must also be relocated to the shadows of the non-vulnerable set. As a result, the total number of shadowed sets decreases, but the size of some shadowed sets increases.

1.3.6 Runtime Support for Leaping Shadow

The performance of Leaping Shadow and its ability to achieve the expected resilience in future extreme-scale computing systems depend heavily of the efficient design and implementation of three main components of the model: (i) A reliable and efficient communication channel between the main process and its associated shadow, (ii) A scalable and robust data structure to efficiently forward messages from mains to shadows, and (iii) Scalable and efficient algorithms to enable *shadow leaping*, with minimum disruption to computing progress.

We intend to explore the potential of native InfiniBand for low latency feature and minimum CPU involvement to support the interaction and communications between a main and its shadow. The objective is to develop a minimal communication layer on top of the InfiniteBand layers, using the channel semantics send and receive primitives, the memory semantics read and write operations, or other RDMA mechanisms, depending on the context of the computation and the state of the main and its shadow. The communication layer, implemented as a library, will support the main communications and leaping shadow functionalities. Multiple challenges must be addressed, however, to enable an efficient and scalable RDMA-enabled Leaping Shadow. In the following, we explore potential research directions and discuss solutions for the design and implementation of this minimal communication layer.

Basic Message Forwarding Structure and Dynamics: Message forwarding between the main and its shadow is necessary to ensure that the shadow reaches a consistent state, after failure. Upon receiving a message from a peer process, the main process asynchronously forwards the message to its shadow. A data structure must, therefore be in place, to efficiently support **persistent asynchronous** communications between the main and its shadow and provide a viable, **intermediate-term storage capacity** for pending messages to be eventually consumed by the shadow. To this end, we will use a **circular message queue** at each end of the RDMA channel to support the forwarding of messages received by the main process from its peers to its associated shadow. At each end of the communication, the message queue will be composed of a set of fixed size buffers. Each buffer in the sender side will be persistently associated with one buffer at the receiver. A head and a tail pointer will be used to ensure consistent, ordered delivery and access of messages in the send and receive queues, respectively. Buffers at the receiving sides can only be reused after the incoming message has been consumed by the shadow task. Therefore, a mechanism must be in place to control the message flow of the queues in the main and shadow side, respectively.

An RDMA-enabled, persistent, asynchronous queue is a viable data structure to support data transfer between the main and its associated shadow, without requiring the involvement of the CPU. The efficiency of such a data structure, however, heavily depends on the semantics of the RDMA channel used. The native InfiniBand Architecture supports both **channel semantics** and **memory semantics**. The channel semantics use the **send** and **receive** operations for communication between the two end points. The sending end of the communication initiates a send operation by posting a send descriptor, which specifies where the source data is located without specifying the destination address at the receiver side. To receive a message, the receiving end of the communication posts a receive descriptor, specifying where the message is to be stored. Upon arrival of the message at the receiver side, the hardware uses the information in the receive descriptor to place data in the destination buffer. Multiple send and receive descriptors can be posted and consumed in FIFO order. A Completion Queue (CQ) is used to notify the application that the operation has completed. In **memory semantics**, two one-sided operations are supported, **RDMA read** and **RDMA write**. The sender initiates an RDMA operation by posting a descriptor, which contains both the local data source address and remote data destination address. Multiple data segments can be specified by a descriptor and the sender is notified of

their completion through the CQs. At the receiver side the operation is served exclusively and transparently by the RDMA transport layer, without involvement of the application layer.

Both InfiniBand semantics offer distinct and valuable advantages to the implementation of Leaping Shadow, but also suffer performance impacting shortcomings. A closer look at the Leaping Shadow dynamics, however, reveals that the interaction between the main and its associated shadow involves long messages, but also short messages, such as heartbeat messages and failure notification messages. Furthermore, the connection can be established at the initialization phase and remains open until either the main fails or the task is completed. So, the use of mechanisms, such as address registration and memory pinning, is no longer of conern. Finally, until failure occurs, the shadow has no interaction with other processes. This eliminates the need to poll other processes for communication.

Based on the above observations, we propose to explore a **hybrid** architecture of the minimal communication layer, which exploits the advantages of the channel and memory semantics of RDMA and avoids their shortcomings, based on the context in which the interaction between the main process and its shadow is taking place. For short data buffers, the InfiniBand channel semantics can be used. Initially, a reliable connection is established between the main and its associated shadow. The CQ mechanism will be used to notify the receiver about incoming messages. Furthermore, the completion of the operations can be detected both at the sender and the receiver side. When large amount of data buffers are involved in the transfer between the main and its shadow, a zero-copy protocol design and implementation will be explored, based on the use of a RDMA writes. We will strive to minimize the number of writes per data buffer transaction, in order to preserve the low latency feature of the InfiniBand RDMA. It is worth noting, however, that since the interaction between the main and its shadow is lasting, the process of pinning down data buffers in memory only occurs when the processes are initialized. Since the main and its shadow will regularly reuse the source and destination buffers, the cost of the initial registration is effectively amortized over the multiple RDMA operations. Furthermore, buffer addresses need only be initially exchanged using control messages. Finally, since the shadow interacts exclusively with the main, the arrival of incoming messages can be detected easily, using polling for example. Similar approaches have been widely used to enhance the performance of MPI over different communication networks. We will leverage this knowledge to achieve efficient and scalable implementation.

Leaping Runtime Support and Execution: Leaping ensures that message buffers are flushed before the message queues overflow their capacity. It provides efficient means by which a leaping shadow achieves forward progress, without execution, using a state that is consistent with that of the main process. Leaping can also be invoked "opportunistically", upon a failure of a main process, to allow the remaining shadows to catch up with their associated mains. The design and implementation of this process requires (i) establishing an efficient notification channel between the main and the shadow to trigger leapind, and (ii) developing efficient mechanisms to update the state of the shadows in a reliable manner to ensure state consistency between the shadow and its main.

The leaping process bears close similarity with the live migration of a virtual machine from one server to another. The main objective of live migration is to achieve migration without excessively halting the virtual machine execution. This typically involves one or more of three phases, a *Push*, a *Stop_and_Copy* and *Pull*. In the Push phase, the source virtual machine transfers its data to the destination virtual machine. In the Stop_and_Copy phase, the source virtual machine stops executing and transfers control to the destination virtual machine. In the Pull phase, the destination retrieves data from the source virtual machines. Different live migration schemes have embedded different phases in their design, with varying levels of success. Several issues, related to the characteristics and dynamics of the leaping process, highlight the difference between this process and live migration. First, in fault-induced leaping, processes must wait until the shadow associated with the failed main reaches the execution point prior to failure. Therefore, stopping is a normal not a forced state that the processes reach. Consequently, no disruption ensues as a result of this process. Secondly, the forced leaping process can be achieved asynchronously, thereby affecting only the main and its associated shadow. Thirdly, depending on the nature of the application and the frequency of synchronization barriers, leaping can be achieved independently and in an uncoordinated fashion between groups of main-shadow pairs. Finally, state update can be achieved incrementally in an efficient manner, since the current state of the shadow is a recent image of a previous state of the main process.

Current live migration schemes do not cope well with the nature of BSP-inspired applications in large-scale envi-

ronments. We will explore new techniques to enable leaping efficiently and at scale. We will leverage the proposed RDMA-enabled minimal layer to achieve reliable communication between the shadow and the main, with minimum delay. More specifically, we will exploit the main features of the native InfiniBand interface to develop and implement efficient techniques to update the CPU execution state of the shadow process and **incrementally** copy the "dirty" pages from the main address space to the shadow's address space. To this end, we will explore hashing-based schemes, with different properties and collision resistance, to produce page fingerprints and identify the dirty pages that must be transmitted. I transfering memory pages containing **page tables**, from the main to the shadow address space, care must be taken to translate the machine dependent addresses to machine independent addresses before the transfer. We will explore resource- and latency-aware, adaptive compression schemes to enhance the page transfer performance of the leaping process, without adverse effects on performance. The main tradeoffs in the design include the effect on the computational resources, the impact on memory and the compression perceived benefit.

1.4 Prototype Implementation

A prototype will be built to experimentally evaluate the performance of Leaping Shadow for two specific applications domains: HPC scientific applications and Internet-scale, data-intensive cloud computing applications. The main objective is to explore the feasibility of Shadow Computing at scale and understand the performance impact of critical design choices of Leaping Shadow on the performance of the application.

To address the first application domain, we will develop the **ls**_MPI] library and implement it as a profiling library on top of OpenMPI. This effort will leverage, SrMPI, a software library developed by our research group, and augment its functionality to support the main components of Leaping Shadow, including shadow overloading, required consistency protocols, message-logging and message-forwarding protocols and state update operations in support of failure-induced and forced leaping [81]. Currently, SrMPI supports replicating MPI ranks, which correspond to tasks in our model, and controlling speed in response to failures. SrMPI supports dynamic execution rate adjustment, which is lacking in existence replication libraries [42, 73, 69]. SrMPI supports a parallel mode of *active* communication consistency, whereby a replica actively processes every request. In active mode, replicas only communicate with the other replica ranks and the main ranks only communicate with other main ranks.

For the second application domain, we will investigate the feasibility of using BSPlib, and develop **ls**_BSP, based on the BSP model of computation [56]. BSP is easy to program, is deadlock free and provides peformance predictability. Its use to develop applications in a very structured manner has been steadily increasing, particularly for cloud computing based applications. We will incorporate the main functionalities of the Leaping Shadow into the BSPlib, and adapt it to a cloud execution environments. We will explore the possibility of augementing BSP-based systems, such as Apache Giraph, an open-souce implementation of Pregle, to enhance their resilience to failure [80].

The main focus of **ls**_BSP implementation would on the allocation of cloud resources to the different computations, and on the integrating the scheduling of tasks and their shadows, to achieve the expected application level resilience. A main challenge that must be addressed in the development of **ls**_BSP is the ability to capture the state of a process and restarting in the same state, thereby allowing the shadow to execute its leap. The structured nature BSP and the simplicity of the superstep abstraction provides a convenient point to capture the global state of the entire BSP computation and migrated it to the shadow address space. Upon transfer of the main state, the shadow process can be restarted to execute the new state.

In order to switch the shadow of the failing main to recovery mode and trigger a failure-induced leaping, failures must be detected. This requires additional "short" messages to be sent either between the main and shadow ranks or to a system to detect node failures. We will explore different protocols to efficiently handle this process, and ensure consistency across all main and shadow processes. In **ls**_MPI, assuming that application non-determinism can only occur during the MPI calls and that the application themselves are deterministic, the only MPI operations that can result in non-deterministic behavior are non-blocking operations, wildcard receive operations, and `MPI_Wtime()`. To handle these cases, we will assume the main rank is always the leader rank and determines the outcome of these operations. If the main node fails then the implementation can simply use the result of the shadow rank. We will also explore and develop similar primitives, based on BSPlib, to support these funcationalities.

When implementing Leaping Shadow's functionalities into OpenMPI or BSPlib, care must be taken to preserve the application's semantics during non-blocking send and receive functions, widely used in MPI applications and collectives. This requires that the Leaping Shadow runtime environment support mapping between the application request pointers and the multiple requests being maintained by the excution environment. To achieve this behavior, we will explore different approaches, focused the modification of the `MPI_Wait()` and `MPI_Test()` functions for **ls**_MPI and the development of semanitically equivalent primitives for **ls**_BSP to translate between the actual request pointers and those handled by application. Special care must be taken to preserve consistency, including the support of wildcard operations.

Within MPI implementations, all communications between ranks happen using a buffer to post both sends and receives. By its very nature, a message is not received until a receive is explicitly posted to a message buffer. This is ideal for Leaping Shadow replication because this property allows messages to be posted to message queues and not actually consumed until the process is ready to receive them. We will take advantage of this property and develop efficient protocols to forward messages from the main to its shadow and control flow between the pair of processes. We will seek to implement similar capabilities in **ls**_BSP, using BSPlib $bsp\_get(), bsp_direct\_get(), bsp\_send(), bsp\_move()$ and $bsp\_qsize()$.

In our implementation, we will incorporate the RDMA-enabled logical channel protocol into Leaping Shadow, to achieve zero-copy efficient data transfer. A concern is that messages must retain their order, particularly when messages are posted to both the main and the shadow ranks. We will make use of an unused tag bit to distinguish between messages received from main and shadows. This allows the receive functions to guarantee order even if messages are interleaved between the main and shadow ranks.

1.5 Related Work

Rollback and recovery are predominate mechanisms to achieve fault tolerance in current HPC environments [37, 61, 100, 25, 61]. Coordinated Checkpoint is a method to achieve a globally consistent state [26]. The major benefit of coordinated checkpointing stems from its simplicity and ease of implementation. Its major drawback, however, a is a lack of scalability, as it requires global process coordination to achieve a checkpoint [39, 44, 97, 20, 23, 53, 90]. In uncoordinated checkpointing, processes record their states independently and postpone creating a globally consistent view until the recovery phase [102, 94]. The major advantage of this scheme is its potential for a reduced overhead during fault free operation [106]. It requires, however, that each process maintain multiple checkpoints and message logs, necessary for recovery. It can also suffer the well-known domino effect, which may result in the re-execution of the entire application [96]. One hybrid approach, known as communication induced checkpointing schemes, aims at reducing coordination overhead, [10, 21]. The approach may cause processes to store useless states that are never used in future rollbacks. To address this shortcoming, "forced checkpoints" have been proposed [55, 67]. This approach may lead to unpredictable checkpointing rates [109, 46]. Incremental checkpointing techniques attempt to address this shortcoming by only writing the changes since the previous checkpoint [27, 7, 92, 93, 38, 7, 27, 72, 90]. These techniques require large memory to support the simultaneous execution of the checkpoint and the application. Multi-level checkpointing, which consists of writing checkpoints to multiple storage targets, has been proposed [87, 99, 52]. The scheme, however, may lead to increased node failure rates. It may also complicate the checkpoint writing process and requires that the system track the current location of all process's checkpoints [28].

Replication was proposed as an alternative to checkpointing in HPC [44, 97, 20, 23, 70, 40, 22, 43]. It has also been proposed, to augment existing checkpointing techniques, and to guard against silent data corruption [44, 23, 88, 101, 36]. The overhead replication can depends strongly on the communication patterns among, but can be as high as 70% [41], particularly when it paired with some form of checkpointing. Efforts have been devoted to cloud computing resilience [60, 108, 103, 104, 65, 111, 89]. These techniques mostly rely on data redundancy and reexecution to enhance Mapreduce-based applications. **To the best of our knowledge,** *leaping shadows* **is the first attempt to explore a state-machine replication based framework that provides a tradeoff between time and hardware redundancy, while meeting resilience and power requirements, with minimum overhead.**

1.6 Broader Impact

This research project will provide students with a unique opportunity to understand the basic science and engineering principles underlying the design of resilient systems, at scale. Research results and methodologies developed as port of this project will be integrated into our courses and seminars on distributed systems and cloud computing. We will also directly involve graduate and undergraduate students in this research program. The CSD at Pitt has a unique program where each faculty has a teaching assistant token that can be dedicated to a graduate student. The token has two semesters of financial support per year. PIs' tokens will be used to support additional students on this project. Given the interdisciplinary nature of the project, the theory/algorithms research community and the computer systems community will be more involved in the green computing revolution. Moreover, the project will create opportunities for theorists and non-theorists to interact. Established theory researcher will figure out how to get involved, advertising to nontheorists what theorists can bring to the table, while systems researcher will provide theorists with new practical problem to investigate and solve.

Students will fully participate in the outreach, dissemination and community effort activities. We will seek to recruit undergraduate students through existing programs at our institutions and through Federal Agencies program funds, including REU supplement and the REU Sites program. The PIs will involve high school students through CS department's Technology Leadership Institute (TLI). The TLI is a multi-week academic enrichment program that builds math and computer science skills, uniquely targeting minority high-school students. The program has been widely recognized and featured in prominent media outlets. This research project will leverage the PITT CS Moye Information Technology Initiative, to target minority and under-represented students, and increase the number of socially, educationally or economically disadvantaged students graduating with CS BS degrees. They will develop and incorporate a summer project in the Moye' initiative. This project will also benefit society and industry. It will bring new insights into the design of energy- and QoS-aware resilient systems, and will provide guidance on how these systems can be implemented at scale to enhance the reliability and availability of next-generation Internet-scale applications, critical to our economy.

1.7 Prior Support

PIs Melhem and Znati collaborated on a 2-year Eager NSF grant (CNS-1252306), which started in 2013, to study scalable resiliency the feasibility of Shadow Computing. The study explored the feasibility and applicabiilty of shadow computing to different computing environments. Four papers resulted from this work [31, 83, 84, 32]. The preliminary results shows that Shadow Computing achieves significant energy savings, while operating under power constraints. The proposed project builds on these results and explores new dimensions of Shadow Computing for emerging computing environments. PIs Znati and Pruhs obtained a grant (CNS-1253218) to explore joint optimization of power management and performance in virtualized environments. The goal is to theoretically investigate balancing energy and QoS in the context of traffic scheduling and routing. In [14] an elegant poly-log-competitive online algorithm were given for the case where the links are speed scalable, and in [66] a poly-log-approximation offline algorithm was given for the case that the nodes are speed scalable.PI Melhem was involved in NSF project CCF-0702452: Compiler and Chip Multiprocessor Co-design for Scalable Efficient Data Access and Communication (Co-PI w/ A. Jones and S. Cho). This project , which was awarded in 2011 and will complete in 2015, takes a cross-layer approach to address cache design and data communication problems in the context of chip multiprocessor systems. It funded 3 PhD students and produced many publications [107, 85, 78, 75, 74, 1, 4, 6, 77, 76, 2, 86, 86, 3, 5]. PI Pruhs obtained two additional NSF grants during the last 5 years; CCF-1421508 - Algorithmic Energy Management in New Information Technologies, and CCF-1115575 - Green Computing Algorithmics. The grants were awarded in 2014 and 2011, respectively, to theoretically investigate balancing energy and QoS when scheduling power-heterogeneous processors. In [48] it was shown that traditional priority based scheduling paradigms could perform badly when scheduling tasks of heterogeneous importance on a power heterogeneous multiprocessor. In [57] a new paradigm was developed for such a situation, and it was shown that the algorithm guarantees a bounded relative error. The key idea is to let jobs migrate selfishly until equilibrium is reached.

Management Plan

The activities of this project will be carried out over a four-year period. The research agenda includes three major thrusts: The first thrust is focused on the theoretical development and analysis of essential algorithms and components of "Shadow Computing". This thrust will analyze the potential of this new paradigm using simplified abstractions for applications and computing systems. The second thrust will remove some of the simplifications required for the feasibility of the first thrust and investigate Shadow Computing in more realistic settings, taking into consideration the implications of communication, sychchronization, realistic reliabilty models and complex objective functions. The third thrust will verify, test and evaluate the techniques developed in the second thrust using simulation and actual implementation for both high performance computing environments (using MPI) and cloud computing environments, using an open source implementation of the BSP model. These thrusts are organized in the following set of tasks:

- Task 1: Develop and analyze a stochastically competitive algorithm for energy minimization when all jobs are initially available.
- Task 2: Develop and analyze a stochastically competitive algorithm for energy minimization with deadline feasibility constraints when jobs arrive over time, or prove that such an algorithm doesn't exist.
- Task 3: Develop and analyze a stochastically competitive algorithm for energy plus total response time when jobs arrive over time, or prove that such an algorithm doesn't exist.
- Task 4: Develop a power-aware optimization framework to model and study energy and power consumption of Leaping Shadows. This framework will be used to minimize energy, meet power constraints and achieve the expected time-to-solution.
- Task 5: Develop power-aware, fault-isolated mapping schemes which use co-locating to reduce execution rate.
- Task 6: Develop power-aware, fault-isolated mapping schemes which use DVFS to reduce execution rate.
- Task 7: Develop a simulation framework to validate the Shadow Computing paradigm and assess the performance of the different heuristics in large-scale HPC environments.
- Task 8: Develop a simulation framework to validate the Shadow Computing paradigm and assess the performance of the different heuristics in large-scale Cloud environments.
- Task 9: Develop a prototype system to experiment with both implementations of Leaping Shadow. We will also carry an experimental study to measure and compare power and energy savings of the mapping- and DVFS-based rate execution control methods, using different classes of applications.

The PIs form a strong research team. Dr. Pruhs is an expert in algorithmic problems related to resource management in general, energy and power management in particular. Dr. Melhem is an expert in parallel, distributed, and high Performance architectures and systems, with a focus on fault-tolerance and QoS reliability. He also made significant contributions to the field of power-aware computing and embedded systems. Dr. Znati is an expert in networking and disributed systems. His work focuses on the design and analysis of highly-reliable and resilient, large-scale systems and applications. All members of the research team will actively particiapte in all aspects of the project, with Dr. Pruhs, Melhem and Znati leading the efforts corresponding to the three thrusts, respectively. The planned schedule of tasks and milestones are shown in Figure 5

The team has a history of successfully working together. Znati and Melhem have collaborated on network management, network security and real-time system. Znati and Pruhs collaborated on energy efficiency in networking and cloud computing infrastructure. The three faculty have jointly served in numerous PhD committees and have written several jointly authored papers. Through this experience, the team has learned how to work smoothly and efficiently together. The investigaors recognize that the project's success depends on a wholly integrated approach. The team is committed to working closely together, as we have done in past projects. Although each PI will participate in all research tasks of the project plan, to ensure focus and quick progress, we have identified particular PIs to lead each one.

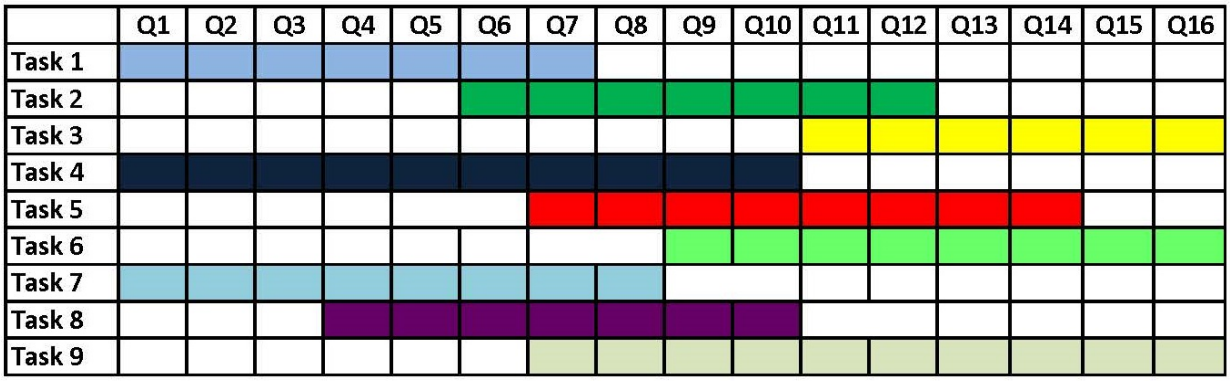


Figure 5: Research Tasks and Milestones.

Pruhs will coordinate the overall effort, to guarantee coherence between the theoretical and system parts of the research. He will also lead the Theory and Algorithm effort. Melhem will lead the framework development and analysis effort. Znati will lead the network and system aspect of the project, effort, and coordinate the simulation and implementation effort.

The PIs are all located at the University of Pittsburgh, which significantly eases project management and collaboration. We already regularly meet and work together, have nearby offices, and collectively advise students. Our past collaborative experience will ensure the project moves forward quickly and smoothly with the emphasis squarely on the research, rather than management issues.

To coordinate the project, we will hold a weekly project meeting with the PIs and students to discuss and resolve research issues, experimental studies, and simulator development. We will also hold a quarterly PI meeting to plan and adjust research progress toward milestones and longer-term directions. We also plan to have a yearly internal workshop where students present their research to other interested groups at the University. To encourage regular collaboration among students, we will locate them in the same lab space, where they can interact and work together on a daily basis.

To share project artifacts within our team, we will use a repository to hold all source code, hardware designs and configurations, internal documents, experimental setups and results, and papers. We anticipate using the OCCAM Experiment Management System (http://www.occamportal.org) being developed at the University of Pittsburgh for experiment archiving. Version control (github) will be used for all software, designs, documents and papers. Additionally, a mailing list, an internal Wiki and a public web site will be created.

References

[1] Ahmed Abousamra, Alex K. Jones, and Rami Melhem. Codesign of noc and cache organization for reducing access latency in chip multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 23:1038–1046, 2012.

[2] Ahmed Abousamra, Alex K. Jones, and Rami Melhem. Ordering circuit establishment in multiplane nocs. *ACM Trans. Des. Autom. Electron. Syst.*, 18(4):49:1–49:33, October 2013.

[3] Ahmed Abousamra, Alex K. Jones, and Rami Melhem. Proactive circuit allocation in multiplane nocs. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 35:1–35:10, New York, NY, USA, 2013. ACM.

[4] Ahmed K. Abousamra, Alex K. Jones, and Rami G. Melhem. Noc-aware cache design for multithreaded execution on tiled chip multiprocessors. In *International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC)*, pages 197–205, 2011.

[5] Ahmed K. Abousamra, Rami G. Melhem, and Alex K. Jones. Déjà vu switching for multiplane nocs. In *Proceedings of the 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, NOCS '12, pages 11–18, Washington, DC, USA, 2012. IEEE Computer Society.

[6] A.K. Abousamra, R.G. Melhem, and A.K. Jones. Déjà vu switching for multiplane nocs. In *IEEE/ACM International Symposium on Networks on Chip (NoCS)*, pages 11 –18, May. 2012.

[7] Saurabh Agarwal, Rahul Garg, Meeta S. Gupta, and Jose E. Moreira. Adaptive incremental checkpointing for massively parallel systems. In *Proceedings of the 18th annual international conference on Supercomputing*, ICS '04, pages 277–286, New York, NY, USA, 2004. ACM.

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

[9] Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration: Extended abstract. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 279–288, 2011.

[10] L. Alvisi, E. Elnozahy, S. Rao, S.A. Husain, and A. de Mel. An analysis of communication induced checkpointing. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 242 –249, 1999.

[11] Lorenzo Alvisi and Keith Marzullp. Message logging: Pessimistic, optimistic, causal, and optimal. *IEEE Trans. Softw. Eng.*, 42(2):149–159, 1998.

[12] Kyoungho An. Resource management and fault tolerance principles for supporting distributed real-time and embedded systems in the cloud. In *Proceedings of the 9th Middleware Doctoral Symposium of the 13th ACM/IFIP/USENIX International Middleware Conference*, MIDDLEWARE '12, pages 4:1–4:6, New York, NY, USA, 2012. ACM.

[13] Antonios Antoniadis, Neal Barcelo, Mario E. Consuegra, Peter Kling, Michael Nugent, Kirk Pruhs, and Michele Scquizzato. Efficient computation of optimal energy and fractional weighted flow trade-off schedules. In *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, pages 63–74, 2014.

[14] Antonios Antoniadis, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, Viswanath Nagarajan, Kirk Pruhs, and Cliff Stein. Hallucination helps: Energy efficient virtual circuit routing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1141–1153, 2014.

[15] Nikhil Bansal, Ho-Leung Chan, Dmitriy Katz, and Kirk Pruhs. Improved bounds for speed scaling in devices obeying the cube-root rule. *Theory of Computing*, 8(1):209–229, 2012.

[16] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. *ACM Transactions on Algorithms*, 9(2):18, 2013.

[17] Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, Kirk Pruhs, and Cliff Stein. Multicast routing for energy minimization using speed scaling. In *Design and Analysis of Algorithms - First Mediterranean Conference on Algorithms, MedAlg 2012, Kibbutz Ein Gedi, Israel, December 3-5, 2012. Proceedings*, pages 37–51, 2012.

[18] Nikhil Bansal and Viswanath Nagarajan. On the adaptivity gap of stochastic orienteering. In Jon Lee and Jens Vygen, editors, *Integer Programming and Combinatorial Optimization*, volume 8494 of *Lecture Notes in Computer Science*, pages 114–125. Springer International Publishing, 2014.

[19] Anand Bhalgat, Ashish Goel, and Sanjeev Khanna. Improved approximation results for stochastic knapsack problems. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1647–1665, 2011.

[20] Marin Bougeret, Henri Casanova, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. Using group replication for resilience on exascale systems. Rapport de recherche RR-7876, INRIA, February 2012.

[21] D. Briatico, A. Ciuffoletti, and L. Simoncini. A Distributed Domino-Effect Free Recovery Algorithm. In *4th IEEE Symposyum on Reliability in Distributed Software and Database Systems*, 1984.

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

[23] Henri Casanova, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. Combining Process Replication and Checkpointing for Resilience on Exascale Systems. Rapport de recherche RR-7951, INRIA, May 2012.

[24] 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. DFTVS '08. IEEE International Symposium on*, pages 114–122, Oct 2008.

[25] K. Mani Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, February 1985.

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

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

[28] Feng Chen, David A. Koufaty, and Xiaodong Zhang. Hystor: Making the best use of solid state drives in high performance storage systems. In *Proceedings of the International Conference on Supercomputing*, ICS '11, pages 22–32, New York, NY, USA, 2011. ACM.

[29] Daniel Cole, Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Speed scaling for stretch plus energy. *Oper. Res. Lett.*, 40(3):180–184, 2012.

[30] Daniel Cole, Dimitrios Letsios, Michael Nugent, and Kirk Pruhs. Optimal energy trade-off schedules. In *2012 International Green Computing Conference, IGCC 2012, San Jose, CA, USA, June 4-8, 2012*, pages 1–10, 2012.

[31] Xiaolong Cui, Bryan Mills, Taieb Znati, and Rami Melhem. Shadow replication: An energy-aware, fault-tolerant computational model for green cloud computing. *Energies*, 7(8):5151–5176, 2014.

[32] Xiaolong Cui, Bryan Mills, Taieb Znati, and Rami Melhem. Shadows on the cloud: An energy-aware, profit maximizing resilience framework for cloud computing. In *International Conference on Cloud Computing and Services Science, CLOSER 2014*, Apr 2014.

[33] Brian C Dean, Michel X Goemans, and Jan Vondrák. Approximating the stochastic knapsack problem: The benefit of adaptivity. *Mathematics of Operations Research*, 33(4):945–964, 2008.

[34] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[35] Jr. E. S. Hertel, R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. PetneY, S. A. Silling, P. A. Taylor, and L. Yarrington. CTH: A software family for multi-dimensional shock physics analysis. In *Proceedings of the 19th International Symposium on Shock Waves*, 1993.

[36] James Elliott, Kishor Kharbas, David Fiala, Frank Mueller, Kurt Ferreira, and Christian Engelmann. Combining partial redundancy and checkpointing for HPC. In *Proceedings of the 2012 IEEE 32Nd International Conference on Distributed Computing Systems*, ICDCS '12, pages 615–626, Washington, DC, USA, 2012. IEEE Computer Society.

[37] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, September 2002.

[38] Elmootazbellah N. Elnozahy and Willy Zwaenepoel. Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit. *IEEE TRANSACTIONS ON COMPUTERS*, 41:526–531, 1992.

[39] E.N. Elnozahy and J.S. Plank. Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery. *Dependable and Secure Computing, IEEE Transactions on*, 1(2):97 – 108, april-june 2004.

[40] C. Engelmann, S.L. Scott, C. Leangsuksun, and X. He. Symmetric active/active high availability for high-performance computing system services: Accomplishments and limitations. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 813–818, May 2008.

[41] Christian Engelmann and Swen Böhm. Redundant execution of HPC applications with MR-MPI. In *Proceedings of the* 10^{th} *IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2011*, pages 31–38, Innsbruck, Austria, February 15-17, 2011. ACTA Press, Calgary, AB, Canada.

[42] Kurt Ferreira, Rolf Riesen, Ron Oldfield, Jon Stearley, James Laros, Kevin Pedretti, and Ron Brightwell. rmpi: increasing fault resiliency in a message-passing environment. Technical Report SAND2011-2488, Sandia National Laboratories, Albuquerque, New Mexico, 2011.

[43] Kurt Ferreira, Jon Stearley, James Laros, Ron Oldfield, Kevin Pedretti, Ron Brightwell, Rolf Riesen, Patrick Bridges, and Dorian Arnold. Evaluating the viability of process replication reliability for exascale systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, New York, NY, USA, 2011. ACM.

[44] Kurt Ferreira, Jon Stearley, James H. Laros, III, Ron Oldfield, Kevin Pedretti, Ron Brightwell, Rolf Riesen, Patrick G. Bridges, and Dorian Arnold. Evaluating the viability of process replication reliability for exascale systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 44:1–44:12, New York, NY, USA, 2011. ACM.

[45] Message P Forum. Mpi: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[46] A. Guermouche, T. Ropars, E. Brunet, M. Snir, and F. Cappello. Uncoordinated checkpointing without domino effect for send-deterministic mpi applications. In *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 989–1000, May 2011.

[47] Sudipto Guha and Kamesh Munagala. Approximation algorithms for budgeted learning problems. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 104–113, New York, NY, USA, 2007. ACM.

[48] Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. Scheduling heterogeneous processors isn't as easy as you think. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1242–1253, 2012.

[49] Anupam Gupta, Ravishankar Krishnaswamy, Marco Molinaro, and R. Ravi. Approximation algorithms for correlated knapsacks and non-martingale bandits. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 827–836, 2011.

[50] Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, and R. Ravi. Approximation algorithms for stochastic orienteering. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1522–1538. SIAM, 2012.

[51] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *Approximation and Online Algorithms - 10th International Workshop, WAOA 2012, Ljubljana, Slovenia, September 13-14, 2012, Revised Selected Papers*, pages 173–186, 2012.

[52] D. Hakkarinen and Zizhong Chen. Multilevel diskless checkpointing. *Computers, IEEE Transactions on*, 62(4):772–783, April 2013.

[53] Paul H Hargrove and Jason C Duell. Berkeley lab checkpoint/restart (blcr) for linux clusters. *Journal of Physics: Conference Series*, 46(1):494, 2006.

[54] P. Hazucha and C. Svensson. Impact of CMOS technology scaling on the atmospheric neutron soft error rate. *Nuclear Science, IEEE Transactions on*, 47(6):2586–2594, Dec 2000.

[55] J.-M. Helary, A. Mostefaoui, R.H.B. Netzer, and M. Raynal. Preventing useless checkpoints in distributed computations. In *Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on*, pages 183 –190, oct 1997.

[56] Jonathan M. D. Hill, Bill Mccoll, Dan C. Stefanescu, Mark W. Goudreau, Kevin Lang, Satish B. Rao, Torsten Suel, Thanasis Tsantilas, and Rob H. Bisseling. Bsplib: The bsp programming library, 1998.

[57] Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. Selfishmigrate: A scalable algorithm for non-clairvoyantly scheduling heterogeneous processors. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 531–540, 2014.

[58] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[59] R. Jhawar, V. Piuri, and M. Santambrogio. Fault tolerance management in cloud computing: a system-level perspective. *IEEE Systems Journal*, 7(2):288–297, June 2013. 1932-8184.

[60] R. Jhawar, V. Piuri, and M. Santambrogio. Fault tolerance management in cloud computing: A system-level perspective. *Systems Journal, IEEE*, 7(2):288–297, 2013.

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

[62] Bala Kalyanasundaram and Kirk Pruhs. Fault-tolerant real-time scheduling. *Algorithmica*, 28(1):125–144, 2000.

[63] Bala Kalyanasundaram and Kirk Pruhs. Fault-tolerant scheduling. *SIAM J. Comput.*, 34(3):697–719, 2005.

[64] Peter Kling and Peter Pietrzyk. Profitable scheduling on multiple speed-scalable processors. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '13, pages 251–260, 2013.

[65] Steven Y. Ko, Imranul Hoque, Brian Cho, and Indranil Gupta. Making cloud intermediate data fault-tolerant. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 181–192, New York, NY, USA, 2010. ACM.

[66] Ravishankar Krishnaswamy, Viswanath Nagarajan, Kirk Pruhs, and Cliff Stein. Cluster before you hallucinate: approximating node-capacitated network design and energy efficient routing. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 734–743, 2014.

[67] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[68] Lawrence Livermore National Laboratory. ASC sequoia benchmark codes, 2013.

[69] Troy LeBlanc, Rakhi Anand, Edgar Gabriel, and Jaspal Subhlok. Volpexmpi: An mpi library for execution of parallel applications on volatile nodes. In Matti Ropo, Jan Westerholm, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 5759 of *Lecture Notes in Computer Science*, pages 124–133. Springer Berlin Heidelberg, 2009.

[70] Arnaud Lefray, Thomas Ropars, and André Schiper. Replication for send-deterministic MPI HPC applications. In *Proceedings of the 3rd Workshop on Fault-tolerance for HPC at Extreme Scale*, FTXS '13, pages 33–40, New York, NY, USA, 2013. ACM.

[71] Jian Li and Wen Yuan. Stochastic combinatorial optimization via poisson approximation. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 971–980, New York, NY, USA, 2013. ACM.

[72] 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, August 1994.

[73] Min Li, Sudharshan S. Vazhkudai, Ali R. Butt, Fei Meng, Xiaosong Ma, Youngjae Kim, Christian Engelmann, and Galen Shipman. Functional partitioning to optimize end-to-end performance on many-core architectures. In *Proceedings of the* 23^{rd} *IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) 2010*, pages 1–12, New Orleans, LA, USA, November 13-19, 2010. ACM Press, New York, NY, USA.

[74] Y. Li, R. Melhem, and A. Jones. Leveraging sharing in second level translation-lookaside buffers for chip multiprocessors. *Computer Architecture Letters*, PP(99):1, 2011.

[75] Yong Li, A. amra, R. Melhem, and A.K. Jones. Compiler-assisted data distribution and network configuration for chip multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 23(11):2058 –2066, Nov. 2012.

[76] Yong Li, R. Melhem, and A.K. Jones. A practical data classification framework for scalable and high performance chip-multiprocessors. *Computers, IEEE Transactions on*, 63(12):2905–2918, Dec 2014.

[77] Yong Li, Rami Melhem, and Alex K. Jones. Practically private: Enabling high performance cmps through compiler-assisted data classification. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 231–240, 2012.

[78] Yong Li, Rami Melhem, and Alex K. Jones. Ps-tlb: Leveraging page classification information for fast, scalable and efficient translation for future cmps. *ACM Trans. Archit. Code Optim.*, 9(4):28:1–28:21, January 2013.

[79] Will Ma. Improvements and generalizations of stochastic knapsack and multi-armed bandit approximation algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1154–1163. SIAM, 2014.

[80] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[81] Bryan Mills. *Power-Aware Resilience for Exascale Computing*. PhD thesis, University of Pittsburgh, Madison, May 2014.

[82] Bryan Mills, Ryan E. Grant, Kurt B. Ferreira, and Rolf Riesen. Evaluating energy saving for checkpoint/restart. In *First International Workshop on Energy Efficient Supercomputing (E2SC) in conjunction with SC13: The International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2013.

[83] Bryan Mills, Taieb Znati, and Rami Melhem. Shadow computing: An energy-aware fault tolerant computing model. In *In Proceedings of the International Conference on Computing, Networking and Communications (ICNC)*, 2014.

[84] Bryan Mills, Taieb Znati, Rami Melhem, Ryan E. Grant, and Kurt 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.

[85] Michael Moeng, Haifeng Xu, Rami Melhem, and Alex K. Jones. Contextprerf: Enhancing the performance and energy of GPUs with non-uniform register access (NURA). *IEEE Transactions on VLSI*, 2014. Conditionally accepted, under revision.

[86] Jones A. Moeng M. and Melhem R. Reciprocal abstraction for computer architecture co-simulation. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2015.

[87] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[88] Xiang Ni, Esteban Meneses, Nikhil Jain, and Laxmikant V. Kalé. Acr: Automatic checkpoint/restart for soft and hard error protection. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 7:1–7:12, New York, NY, USA, 2013. ACM.

[89] Bogdan Nicolae and Franck Cappello. Blobcr: efficient checkpoint-restart for hpc applications on iaas clouds using virtual disk image snapshots. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 34:1–34:12, New York, NY, USA, 2011. ACM.

[90] 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 *Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on*, pages 30 –46, sept. 2007.

[91] Prasenjit Kumar Patra, Harshpreet Singh, and Gurpreet Singh. Article: Fault tolerance techniques and comparative implementation in cloud computing. *International Journal of Computer Applications*, 64(14):37–41, February 2013. Full text available.

[92] James S. Plank, Youngbae Kim, and Jack J. Dongarra. Algorithm-based diskless checkpointing for fault-tolerant matrix operations. In *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing*, FTCS '95, page 351, Washington, DC, USA, 1995. IEEE Computer Society.

[93] J.S. Plank and Kai Li. Faster checkpointing with n+1 parity. In *Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*, pages 288–297, June 1994.

[94] J.S. Plank and M.G. Thomason. The average availability of parallel checkpointing systems and its importance in selecting runtime parameters. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 250 –257, 1999.

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

[96] B. Randell. System structure for software fault tolerance. In *Proceedings of the international conference on Reliable software*, pages 437–449, New York, NY, USA, 1975. ACM.

[97] Rolf Riesen, Kurt Ferreira, Jon R. Stearley, Ron Oldfield, James H. Laros III, Kevin T. Pedretti, and Ron Brightwell. Redundant computing for exascale systems. (SAND2010-8709), December 2010.

[98] Sandia National Laboratory. Mantevo project home page, 2010.

[99] Kento Sato, Naoya Maruyama, Kathryn Mohror, Adam Moody, Todd Gamblin, Bronis R. de Supinski, and Satoshi Matsuoka. Design and modeling of a non-blocking checkpointing system. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 19:1–19:10, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[100] D.P. Siewiorek and R.S. Swarz. *The theory and practice of reliable system design*. Digital Press, 1982.

[101] J. Stearley, K. Ferreira, D. Robinson, J. Laros, K. Pedretti, D. Arnold, P. Bridges, and R. Riesen. Does partial replication pay off? In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–6, June 2012.

[102] Rob Strom and Shaula Yemini. Optimistic recovery in distributed systems. *ACM Trans. Comput. Syst.*, 3(3):204–226, August 1985.

[103] A. Tchana, L. Broto, and D. Hagimont. Approaches to cloud computing fault tolerance. In *Computer, Information and Telecommunication Systems (CITS), 2012 International Conference on*, pages 1–6, 2012.

[104] W.-T. Tsai, Peide Zhong, J. Elston, Xiaoying Bai, and Yinong Chen. Service replication strategies with mapreduce in clouds. In *Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on*, pages 381–388, 2011.

[105] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[106] Yi-Min Wang, Yennun Huang, and W.K. Fuchs. Progressive retry for software error recovery in distributed systems. In *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, pages 138–144, June 1993.

[107] Haifeng Xu, Rami Melhem, and Alex K. Jones. Multilane racetrack caches: Improving efficiency through compression and independent shifting. In *in Proceedings of the 2015 Asia, South Pacific Design Automation Conference (ASPDAC)*, 2015.

[108] Wenbing Zhao, P.M. Melliar-Smith, and L.E. Moser. Fault tolerance middleware for cloud computing. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 67–74, 2010.

[109] Gengbin Zheng, Lixia Shi, and L.V. Kale. FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In *Cluster Computing, 2004 IEEE International Conference on*, pages 93–103, Sept 2004.

[110] Qin Zheng. Improving mapreduce fault tolerance in the cloud. In *IPDPS Workshops*, pages 1–6. IEEE, 2010.

[111] Qin 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, 2010.

[112] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 35–40, 2004.