Lazy Shadowing - An Adaptive, Power-Aware, Resiliency Framework for Extreme Scale Computing

R. Melhem, University of Pittsburgh (Principal Investigator)
T. Znati, University of Pittsburgh (Co-Investigator)
E. Meneses, University of Pittsburgh (Co-Investigator)
K. Kant, Temple University (Co-Investigator)

Street address:
210 South Bouquet St
Pittsburgh, PA 15213

Administrative point of Contact:
Jennifer E. Woodward, PhD, Associate Vice Provost for Research Operations,
Office of Research
University of Pittsburgh
123 University Place, B21
Pittsburgh, PA 15213-2303

Funding opportunity FOA Number: DE-FOA-0001059

DOE/Office of Science Advanced Scientific Computing Research

DOE/Office of Science Advanced Scientific Computing Research Technical Contact: Dr. Lucy Nowell

PAMS Pre-application tracking number: PRE-0000004476

Research area: Fault Mitigation

Collaborative Application Information

	Names	Institution	Year 1 Budget	Year 2 Budget	Year 3 Budget	total Budget
Lead PI	Rami Melhem	U. of Pittsburgh	$327,701	$337,070	$346,720	$1,011,492
Co-PI	Krishna Kant	Temple University	$75,000	$75,000	$75,000	$225,000

The lead PI for this Project is Rami Melhem. He will be the main point of contact for the project.

The lead PI will coordinate the project's activities within the University of Pittsburgh and between the University of Pittsburgh and Temple University. Graduate students at the University of Pittsburgh will be co-advised by Professors Melhem and Znati and graduate students at Temple will be advised by Professor Kant.

1 Background and Motivation

To enable future scientific breakthroughs and discoveries, the next generation of scientific applications will require exascale computing performance to support the execution of predictive models and analysis of massive quantities of data, with significantly higher resolution and fidelity than what is possible within existing computing infrastructure. In order to achieve the desired functional performance of these applications, several daunting scalability challenges must be addressed. In the late 90's, terascale performance was achieved with fewer than 10,000 heavyweight single-core processors. A decade later, petascale performance required about ten times as many processors as terascale performance. Delivering exascale computing will require one million processors, each supporting 1,000 cores, resulting in a billion-core computing infrastructure while also requiring a dramatic increase in the number of memory modules, communications devices and storage components, as depicted in Table 1. In this table, the MTBF is 5 years and the checkpoint time is 20 minutes, which is estimated based on a system memory of (32-64PB) and an I/O bandwidth of (20-60TB) projected in the "swim-lanes". The reported data was produced using a simulator [56].

The large increase in number of components significantly increases the propensity of exascale computing systems to faults, while driving power consumption to unforeseen heights. Unfortunately, in so far as performance is concerned, resilience to failures and adherence to power budget constraints are two conflicting objectives, as achieving high performance may push the system's components past their thermal limit and increase their likelihood of failure.

With the explosive growth in the number of components comes a dramatic increase in system power requirements. Figure 1(a) shows 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, making system power a leading design constraint on the path to exascale computing. DoE has recognized this trend and established a power limit of 20 megawatt [2], challenging the research community to provide a 1000x improvement in performance with only a 10x increase in power.

		Exascale (projection)		
System Parameter	Petascale	Swim Lane 1	Swim Lane 2	Factor Change
System Peak	2Pf/s	1 Ef/s		500
Power	6MW	$\leq$ 20MW		3
System Memory	0.3 PB	32 - 64 PB		100-200
Total Concurrency	225K	$1B \times 10$	$1B \times 100$	40,000-400,000
Node Performance	125GF	1TF	10TF	8-80
Node Concurrency	12	1,000	10,000	83-830
Network BW	1.5 GB/s	100 GB/s	1,000 GB/s	66-660
System Size (nodes)	18,700	1,000,000	100,000	50-500
I/O Capacity	15 PB	32 - 64 PB		20-67
I/O BW	0.2 TB/s	20-60 TB/s		10-30

Table 1: Comparison of a petascale supercomputer to an expected exascale-class supercomputer (DoE Exascale Workshop [2]).

Although largely pragmatic[1], adhereing to a prescribed power constraints is critical to the design and operations of exascale computing systems. The power-limits placed upon exascale-class systems is at the heart of the tradeoff between power consumption and time-to-solution. Although the intricacies of this tradeoff are complex, operating under power-constraints is a reality that future exascale system designers must face, making the relationship between computing performance and power consumption increasingly more critical.

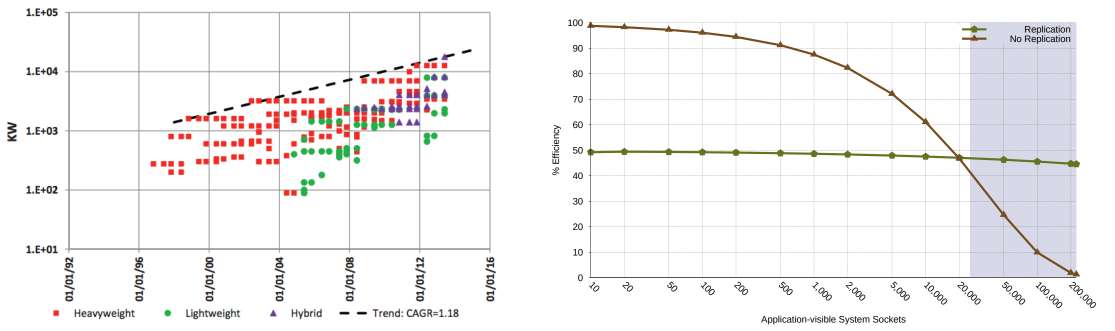
Another direct consequence of the increase in the number of components, is the inreasing propencity of the system to failure. Regardless of the individual components' reliability, system resilience will continue to decrease as the number of components increases[2]. To avoid the full re-execution of a failing application, traditional fault-tolerant techniques typically checkpoint the execution state of the applications, periodically. Upon the occurrence of a fault, recovery is achieved by restarting the computation from a safe checkpoint [34]. Recent work [46, 45, 28, 55, 8, 11] have shown that existing solutions are likely not to scale to the level of faults anticipated in exascale environments. Given the increase in system failure rates and the time required to checkpoint large-scale compute- and data-intensive applications, it is very likely that the time required to periodically checkpoint an application and restart it upon failure may exceed the system mean time between failures [55]. Consequently, applications may achieve very little computing progress, thereby reducing considerably the overall performance of the system. Furthermore, the nature and frequency of errors in exascale computing are such that checkpoint-restart may not be an adequate approach to handle the diverse types of faults in these environments. Figure 1 illustrates this fact and shows that, using coordinated checkpointing, the system efficiency drops below 50% as the number of sockets increases. Several studies have shown this same behavior and proposed traditional replication as a possible solution [49, 55].

Replication, also referred to as state-machine replication, is a well-known technique that has been shown to scale to

[1] The constraint is derived from an estimated cost of $1 million per MW per year, resulting in a power budget of $20 million per year.
[2] For example, a system of 1 million components, each averaging a fault every 25 years, is expected to experience a fault every 10 minutes.

meet the resilience requirements of large distributed and mission-critical systems. Based on this technique, a processes state and computation are replicated across independent computing nodes. When the main process fails, one of the replicas takes over the computation task. Replication requires doubling the number of nodes, since each process must have at least 1 replica, thereby reducing the system efficiency to 50%. A major shortcoming of traditional replication in HPC is the increased power consumption caused by the need for additional resources to tolerate failure, which might exceed the power budget imposed upon exascale-class machines.

The inherent instability of exascale computing systems, in terms of the envisioned high-rate and diversity of their faults, and the challenging power constraints under which these systems will be designed to operate, calls for a reconsideration of the fault tolerance problem. The main objective of the proposed research project is to explore novel paradigms for resiliency in future exascale systems. To this end, we propose a proactive, power-aware resiliency model, referred to as *Lazy Shadowing*, as an efficient and scalable alternative to achieve high-levels of resiliency, through forward progress, in extreme-scale, failure-prone computing environments. In the proposed resiliency model, each process is associated with a **shadow**, a **derivative** of the main process, which runs concurrently with the main process. The shadow executes the same code as its associated main process, but at a **reduced CPU rate** and on a **different** computing node. The rate at which the shadow runs is derived based on the expected time-to-solution of the supported application, the power constraints imposed by the underlying computing infrastructuer and the likelihood of failure. The successful completion of the main process, however, results in the immediate termination of the shadow. Upon failure of the main process, however, the shadow increases its execution rate to complete the task, thereby reducing the impact of such a failure on the progress of the remaining tasks. Since the failure of an individual component is much lower than the overall system failure, it is very likely that, most of the main processes complete their execution successfully. Consequently, the coupled execution of a main process and its associated shadow dramatically increases a power-constrained system's tolerance to failure, at a significantly reduced energy conmption.



(a) System power projections [35]

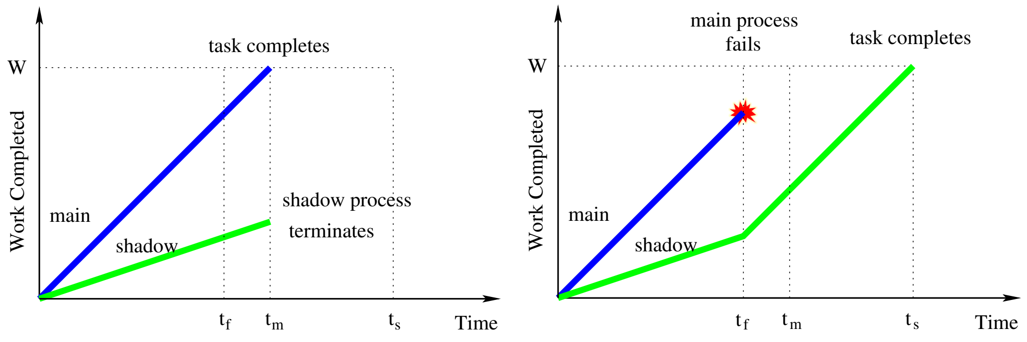
(b) Effect on System Efficiency as the number of nodes increases.

Figure 1: Power projection and system efficiency.

1.1 Objective and Research Thrusts

There is a direct relationship between functional and application performance, although achieving application performance through functional performance may prove to be challenging in failure prone environments [6]. The main design goal of *Lazy Shadowing* is to improve functional performance, by incresing the tolerance of processes to failure, thereby enabling applications to efficiently use computational, energy and time resources, while adhering to power constraints. To achieve this goal, the focus of the resarch agenda will be on the following areas:

- **Understanding Lazy Shadowing at Scale**: We will develop analytical and optimization models to gain better understanding of the behavior and performance of the basic *Lazy Shadowing* concept, for different failure models and different classes of scientific applications. We will explore different approaches to reduce the shadow's execution rate and study the potential of *Lazy Shadowing* to achieve high-levels of energy saving while meeting the power constraints of the underlying computational infratructure. We will also develop a comprehensive power-model to compute the expected energy consumption of *Lazy Shadowing* and an optimization framework to determine the energy-optimal execution rates of the main and shadow processes, for different application requirements.
- **Jumping Shadow for Extreme Scale Computing**: Armed with the outcome of the previous thrust, we will focus on a particular type of *Lazy Shadowing*, referred to as *Jumping Shadow*. Unlike in checkpointing, recovery



(a) The main does not fail. (b) The shadow executes faster after its main fails.

Figure 2: The Lazy Shadowing Computational Model.

and state restoration in Jumping Shadow overlap, resulting in improved performance and energy saving. We will use simulation to explore the performance of *Jumping Shadow* for different failure models and classes of applications, operating under power constraints. We will investigate *Shadow Overloading* as well as *DVFS* as mechanisms to reduce the *Jumping Shadow*'s execution rate, and explore differnt mapping of shadows to computing cores to achieve high-energy efficiency, while achieving the expected levels of resilience.

- **Runtime Support for *Jumping Shadow***: In this thrust, we will develop RDMA-enabled network channels to efficiently support the runtime and communiacations requirements of *Jumping Shadow*, focusing on scale and efficiency. While the use of RDMA-enabled channels has been proposed before, its application to *Jumping Shadow* requires addressing several issues inherent to the dynamics of this resilience approach. We will also investigate efficient data structures to support flow-controlled message forwarding and event notification between mains and their shadows.
- **Proof-of-Concept Prototype for *Jumping Shadow***: To assess the feasibility of *Jumping Shadow* as a resilience model, we will develop a prototype, that leverages several existing technologies to integrate *Lazy Shadowing* functionalaties into MPI. This integration will enable a **non-modified** MPI application to execute in HPC systems, while taking advantage of the Jumping Shadow resiliency features. We will also develop an experimental study to assess the energy saving and resilience of *Jumping Shadow*.

2 Lazy Shadowing: Road map for Resilience at Scale

Enabling *Lazy Shadowing* for resiliency in extreme scale computing brings about a number of challenges and design decisions that need to be addressed, including the applicability of this concept to a large number of tasks executing in parallel, the runtime mechanisms and communications support to ensure efficient interaction between a main and its shadow and the non-intrusive integration of the resiliency in MPI. In the following, we first discuss how the basic concept of *Lazy Shadowing* is applied to multiple tasks executing in parallel. We then propose, *Jumping Shadow* as an efficient technique to ensure forward progress of all shadows of non-failing mains. We also discuss the runtime design issues related to enabling runtime support to efficiently achieve the expected levels of resilience in exascale systems.

2.1 Shadowing a simple task

The basic tenet of the proposed resilience model is the concept of shadowing, whereby each process is shadowed by a *lazy* replica The basic execution dynamics of lazy shadowing, in the case of a single task of size W, is depicted in Figure 2. Both the main process and its shadow simultaneously start executing, the main at a rate σ_m and the lazy shadow at a lower rate $\sigma_s \leq \sigma_m$. If the main process does not fail, it completes its execution at time $t_m = W/\sigma_m$, which causes the immediate termination of the shadow, as illustrated in Figure 2(a). However, if at time t_f, the main process fails, the shadow increases its execution rate to σ_a to complete the task at time $t_s = t_f + (W - \sigma_s * t_f)/\sigma_a$, as depicted in Figure 2(b). More specifically, the shadow completes $\sigma_s * t_f$ work by time t_f, and completes the remaining $(W - \sigma_s * t_f)$ work at a rate σ_a. The expected completion time, T, of the task can be easily computed by noting that t_s is a function of t_f and integrating $t_s(t_f) * f(t_f)$, where $f(t_f)$ is the probability of a fault occurring at time t_f, taking into consideration the probability that the main will complete at time t_m if no failure occurs.

The energy consumed during execution is composed of a static energy, which is proportional to the task's completion time, and a dynamic energy, which depends on the actual executed work. If voltage and frequency scaling (DVFS) is used to control the execution rates, then the dynamic power consumed when a process executes at a speed σ is proportional to σ^{α}, where α is between 2 and 3. The expected energy consumption can be obtained by computing the integral of the energy consumption of the mains and shadows when a fault occurs at time t_f with probability $f(t_f)$.

Clearly, for a given task size, W, each of the expected execution time, T, the expected power consumption, P, and the expected energy consumption, E, is a function of the execution rates, σ_m, σ_s and σ_a. Hence, for a given failure probability distribution, these speeds can be carefully chosen to minimize any of T, P or E, to niminize T given constraints on E or P, or to optimize a cost metric expressed as a combination of T, P and E. In HPC, however, throughput consideration requires that the rates of the main task, σ_m, and the shadow after failure, σ_a, be set to the maximum execution rate. The execution rate of the shadow before failure, σ_s, however, may still be used to manage the trade-offs between T, P and E discussed above. Smaller σ_s corresponds to lazier shadowing of the main process.

2.2 Shadowing multiple tasks

Assume that a job is distributed to N tasks and executed in parallel on N cores. If each of the N tasks is to be shadowed, then it is possible to use $2N$ cores for execution, N to execute the main tasks at the maximum rate and N to *lazily* execute the shadows at a fraction, σ_s, of the maximum rate using DVFS. Alternatively, it is possible to use $N+K$ cores, where N is a multiple of K, and colocate N/K shadows on each of the K cores, while executing all the cores at the maximum speed. Collocating shadows effectively limits the maximum execution rate of each shadow to $\sigma_s = K/N$. 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.

In this project, we will consider both alternatives for lazy shadowing. However, due to space limitation, we will focus the rest of our discussion of the main ideas, concepts and design on shadow colocating. 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.

HPC systems are usually built using multiple nodes, each consisting of multiple sockets with multiple cores in each socket. A fault may affect a core, a socket or a node, and this should determine the mapping of shadowed sets into the actual architecture. Specifically, this 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 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), groups $\{M(0), M(1)\}$ 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 groups are vulnerable while only groups $\{M(12), M(13), S(12,13)\}$ 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) while it has to support twelve processes (shadows) in the third socket. Increased memory pressure on the socket executing the twelve shadows will translate to potentially more

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.

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

Figure 4: An example of socket level shadowing.

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.

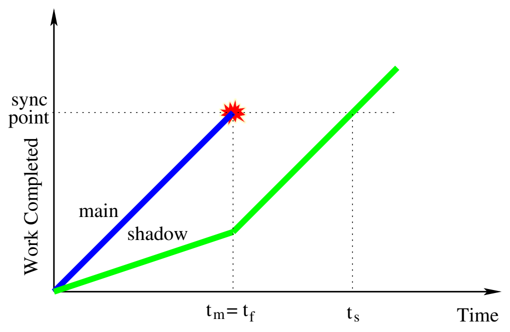
2.3 Dealing with communicating tasks

The communication model used in Shadow Computing assumes that the underlying hardware consists in a collection of cores connected through a high-speed network. A specific number ($N+K$ or $2N$, depending on the shadow mechanism employed) is used to run an application with N tasks. Each pair of cores running a task and its shadow are conceptually connected through a channel that respects First-in-first-out (FIFO) ordering. Therefore, two messages between the same source and destination can never be reordered. Note that communicating tasks are an essential part of Shadow Computing giving the prominence of message-passing programs in the portfolio of the Department of Energy. The Message Passing Interface (MPI) is by far the dominant paradigm to write parallel HPC applications.

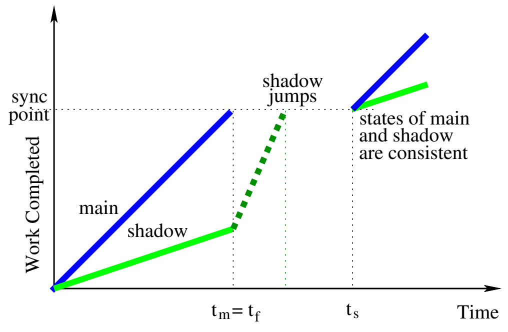
An application using lazy shadowing requires an infrastructure which ensures that any execution (even in the presence of failures) finishes with correct results. That includes: a) a protocol to keep a shadow task consistent with its main task, b) a mechanism to store the necessary amount of messages, c) a fault detection and notification mechanism and d) a recovery method that ensures the system reaches a consistent state after a failure.

The fundamental problem to address is to prevent a main M(i) and its shadow S(i) from diverging in their states. Shadow S(i) will typically lag behind M(i) during execution, so it is necessary to ensure that S(i) reaches the same states as M(i) in case of a failure. We will use a message-logging protocol [4] to satisfy that requirement. This type of 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 a few 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, shadow S(i) may deviate from the state at M(i) (keep in mind that floating-point arithmetic is non associative). 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. In addition, we must highlight that shadows are mute in the sense that all outgoing messages from shadows are suppressed.

To provide correct recovery after failure, a mechanism is required to guarantee that the system reaches a consistent state after recovery, i.e., message queues are flushed, missing messages are replayed, and there are no data structures in intermediate states. After a main task M(i) fails, the failure notification mechanism will inform all the tasks in the system and S(i) will then take over M(i)'s role. If there are other shadows sharing the same core with S(i), they will be terminated and S(i) will start consuming the messages in its receiver-side message log at a faster speed. The



(a) The shadow of the faulty main recovers.



(b) The shadows of non-faulty mains roll forward.

Figure 5: The concept of Jumping Shadows.

message logging protocol will ensure that shadow S(i), now a main task, reaches a consistent state with the rest of the system. The other shadows will roll forward (jump) 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.

Because a shadow task executes slower than its main, the size of the message queues is likely to continue to grow (message creation is faster than its consumption). Consequently, one research problem is to determine an efficient strategy to reduce the size of the receiver-side message log. We will use the strategy of forced shadow jumping that will be described in the next section and exploit application patterns to avoid storing all messages.

2.4 Failure-induced shadow jumping for heavily synchronized tasks

The same optimizations and trade-offs between performance, power and energy discussed for the lazy shadowing of a single task can be applied in the case of N tasks, M(0), ..., M($N-1$) and N shadows, S(0), ... , S($N-1$), as as long as the tasks execute independently without synchronization. If execution is divided into phases with synchronization only between phases, then a global optimization may be also formulated by applying a local optimization within each phase. The behavior within one phase in which each task executes W units of work is still captured by Figure 2. Specifically, with N tasks, Figure 2(b) illustrates the behavior of any main/shadow pair when the main fails at time t_f at which time the shadow executes at maximum rate. Figure 2(a) illustrates the behavior of all other main/shadow pairs that continue execution, not necessarily aware of the failure. However, because the tasks have to synchronize after each one executes W units of work, then all the mains that did not fail will reach that synchronization point at time t_m while the shadow of the failed main will reach that point at the later time, t_s. Hence, the non-faulty mains will be idle from t_m until t_s.

Clearly, the inefficiency implied by the non-faulty mains waiting for the shadow of a faulty main at a synchronization point becomes more acute when synchronization points are frequent, which is the case in many HPC applications. Hence, we will concentrate in the rest of this proposal on shadowing for tasks that synchronize very frequently. Specifically, we will assume that time between synchronizations is very small compared to the total execution time. Consequently, if one main, M(i), fails at time t_f, then each other mains will have to wait for S(i) very soon after t_f (at time $t_m = t_f + \varepsilon$, where $\varepsilon \rightarrow 0$).

The waiting time, $t_w = t_s - t_m = (1-\sigma_s)t_f$, is caused by the fact that $\sigma_s < 1$. To utilize the waiting time between t_m and t_s effectively, we propose to use this time to align the state of the shadows with those of the corresponding mains. We call this technique **Jumping Shadows**. Specifically, for the shadows corresponding to non-failing mains, the state of each shadow is updated by rolling forward to the state of its main. This way, after the synchronization point, the main and its shadow will resume execution from the same consistent state (Figure 5). Clearly the waiting time, t_w increases with t_f and S. The time for the S shadows in a shadowed set to update their states (jump) to match the states of their corresponding mains increases with S, and with the difference between the states of the mains and shadows, which increases with t_f. We will study this relation thoroughly, keeping in mind that the size of shadowed group, $S+1$, may be constrained by the requirement that all S shadows in a group can jump within the period, t_w.

2.5 Forced shadow jumping for efficient logging

Jumping shadows induced by a fault release the buffer storage dedicated for holding the messages that will be consumed by the slow shadows. However, in the absence of faults, the size of the message buffers continously increases. One method for working with limited buffer capacities is to set a lower bound on the speed of the shadows. This bound forces the shadows to consume messages at a rate that prevents buffer overflow. Another approach is to force the shadows to jump and catch up with their mains when the buffer utilization reaches a certain threashold. We call this forced shadow jumping, which applies a mechanism similar to failure induced jumping, but is invoked if no failure occurs for an extended period of time. We will discuss possible implementations of both fault-induced jumping and forced jumping in Section 3.

To evaluate the buffer size constraints, we ran a number of representative HPC workloads and used the rMPI replication library to collect the mean message logging growth rate of the application. These workloads include the production applications CTH [18], a shock physics code, LAMMPS [53], the molecular dynamic code, and the algebraic multi-grid solver AMG [39]. We also include results from two of the mini-applications from Sandia's mantevo suite [57]: HPCCG (a conjugate gradient solver) and miniFE (an implicit finite element method). These applications represent a range of computational techniques, which are frequently run at very large scales on leadership class systems, and represent key simulation workloads for the U. S. DoE. We measured the maximum message log growth rate for each application, shown in Table 2. From this table, we see 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 messge rate.

Application	Message Rate(Bps)
CTH	5.42×10^7
miniFE	2.58×10^6
LAMMPS	2.16×10^6
AMG	9.66×10^5
HPCCG	7.43×10^5

Table 2: HPC workloads maximum message rates.

2.6 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. In Figure 6, we show a possible rejuvenated configuration assuming that the failure of the cores executing $M(1)$ and $M(14)$ in Figure 3(b) is permanent. In this restored configuration, the number of shadowed sets is reduced from 8 to 6, with four sets containing three mains each and two sets containing two mains each. Rejuvenating the vulnerable configuration of Figure 3(c) after a permanent socket failure is more complex but follows the same basic principle.

2.7 Preliminary Results

A preliminary framework was developed to explore the potential and feasibility of *Lazy Shadowing* in dealing with failures, and compare its performance to coordinated checkpointing and rollback recovery, typically used in

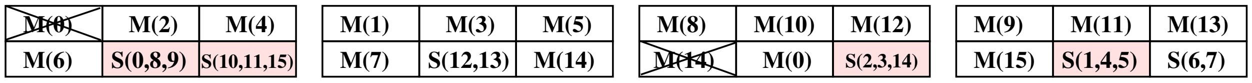


Figure 6: Shadowed set rejuvenation of the vulnerable sets resulting from permanent faults.

Socket Exepected Energy	$E_{soc}(\sigma,[t_1,t_2]) \quad = \quad \int_{t=t_1}^{t_2}(\sigma^3+\rho\sigma_{max}^3)dt$	(1)
I/O Expected Energy	$E_{io}([t_1,t_2]) = \int_{t=t_1}^{t_2}(\gamma\,\sigma_{max}^3)dt = (\gamma\,\sigma_{max}^3)(t_2-t_1)$	(2)
Checkpointing Expected Time	$T_w = M_{sys}e^{R/M_{sys}}(e^{(\tau+\delta)/M_{sys}}-1)(\frac{T_s}{\tau}-\frac{\delta}{\tau+\delta})$	(3)
Checkpointing Expected Energy	$E_{cpr} = E_{io}([0,\delta)]) \times \frac{T_s}{\tau} + E_{io}([0,R)]) \times \frac{T_w}{M_{sys}} + E_{soc}(\sigma_{max},[0,T_w])$	(4)
Lazy Shadowing Expected Energy	$E_{shd} = \int_{t=0}^{t_m}(E_{soc}(\sigma_{max},[0,t]) + E_{soc}(\sigma_s,[0,t]))f(t)dt$ $+\int_{t=0}^{t_m}E_{soc}(\sigma_a,[t,t_r])f(t)dt$ $+(1-\int_{t=0}^{t_m}f(t)dt)(E_{soc}(\sigma_{max},[0,t_m]) + E_{soc}(\sigma_s,[0,t_m]))$	(5)

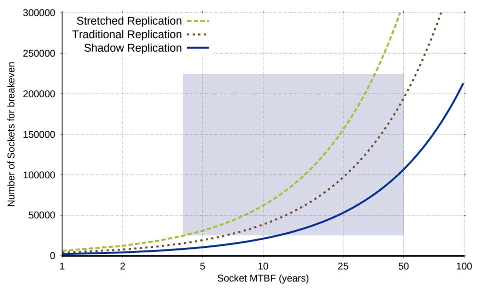
Table 3: Checkpointing and Lazy Shadowing Energy Consumption

high-performance computing. The study focuses on evaluating the ability of the schemes to tolerate failures, while achieving the same throughput and operating under the same power thresholds. We consider a distributed computing environment runing a large number of tasks, N, equivalent to ranks in MPI, to complete an application. The successful execution of the application depends on the successful completion of all tasks. Therefore, the failure of a single task delays the entire application. We assume the application to be perfectly parallelizable and has strong scaling; therefore, as the number of sockets increases, the amount of work each individual process performs, W_{task}, decreases. In this preliminary study, DVFS is used to achieve a lower execution speed, σ, less than the maximum speed, σ_{max}. We further assume that failures are independent events, that can occur at any point during execution according to a failure probablity density function $f(t) = \frac{1}{M_{soc}}e^{-t/M_{soc}}$, where M_{soc} is the socket MTBF.

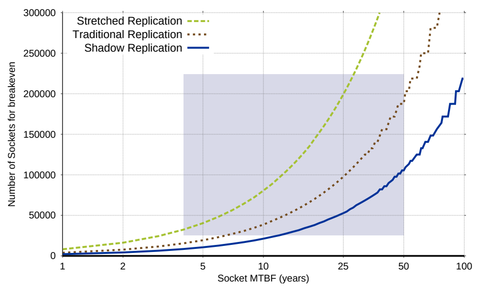
Dynamic power is computed as $P(\sigma) = \sigma^3$ and static power is estimated to include both CPU leakage and other components consuming power during execution, such as memory and network and storage devices. We assume that the static power is a fixed factor, ρ, of the total power consumed at full speed. Based on the above model, the energy consumed by a socket executing at speed σ during an interval $[t_1,t_2]$ is given by Equation (1) in Table 3. We also consider the energy consumed during an Input/Output operation for writing or recovering a checkpoint. We handle this case separately because studies have shown that this operation consumes a significant amount of energy [19]. Similar to static power, we define maximum I/O power as a factor, γ, of the CPU when operating at full speed. Consequently, the energy consumed by I/O operations over an interval $[t_1,t_2]$ is given by Equation (2) in Table 3.

The computational, power and failure models, described above, are used to compare the performance of Coordinated Checkpointing and *Lazy Shadowing*. Coordinated checkpointing periodically preempts task execution to save a checkpoint. If any one socket fails then this checkpoint is used to restart execution. Daly [17] computes the expected total wall clock time, T_w, as Equation (3) in Table 3, given the original total solution time (T_s), a system MTBF (M_{sys}), checkpoint interval (τ), checkpoint time (δ) and recovery time (R). Based on the above, the estimated energy, E_{cpr}, required for a single process, using checkpoint and restart (CPR), is given by Equation (4) in Table 3. To compute the energy consumed by N sockets, E_{cpr} is multiplied by N.

To compute the energy consumed by *Lazy Shadowing*, we focus on the dynamics of the execution of main and its associated shadow, as shown in Figure 2. Recall that the shadow process executes at two different speeds, a speed before failure detection, σ_s, and a speed after failure detection, σ_a. We define the expected energy of a *Lazy Shadowing* pair as the summation of the expected energy consumed by the main and shadow process given our failure model. The expected energy, E_{shd} is given by Equation (5) in Table 3. The first part of the expression is the expected energy consumed by the main and shadow process before a failure occurs. The second part is the expected energy consumed by the shadow after failure occurs. The last part is the expected energy consumed by the main and shadow processes in the event that no failure occurs. This equation is used as an objective function of an optimization problem to find shadow execution speeds. A simplified version of this problem assumes that the speed of the shadow after failure is $\sigma_a = \sigma_{max}$. This assumptions reduces the original optimization problem to finding the speed of the shadow process before failure, σ_s. Taking the derivative of E_{rep} and setting it equal to 0, leads to the value of σ_s. Note that Equation



(a) Breakeven Energy



(b) Breakeven Time

Figure 7: Energy and time breakeven for a fixed checkpoint time of 15 minutes and a power overhead of 60%. Exascale systems are expected to fall into the shaded region.

(5) can also be used to represent traditional replication and *stretched replication*, by setting $\sigma_m = \sigma_s = \sigma_a = \sigma_{max}$, and $\sigma_m = \sigma_s = \sigma_a = \frac{W_{task}}{T_w}$, respectively.

Performance Results: We compare fault tolerance efficiencies by identifying the breakeven point at which the *Lazy Shadowing* technique is equivalent to that provided by coordinated checkpointing. We use two different breakeven metrics, the expected energy consumed and the expected time to solution. Breakeven values are the points at which the energy (or time) required for coordinated checkpointing is equal to the energy (or time) required for *Lazy Shadowing* . Figure 7(a) shows the energy breakeven point for various system sizes and socket MTBF using a fixed checkpoint time of 15 minutes. The area above the breakeven curve is where *Lazy Shadowing* is more efficient than coordinated checkpointing. The results show that *Lazy Shadowing* can provide a significant energy savings over traditional replication. For example when socket MTBF is 25 years traditional replication is viable at 96,700 sockets, whereas *Lazy Shadowing* is more efficient at 53,100. This represents a 46% energy efficiency gain. Stretched replication is the less energy efficient because of the increased time to solution and the presence of overhead power.

Lazy Shadowing achieves this energy savings by slowing down the replicas, which raises the question of how this strategy affects the expected time to completion. Figure 7(b) plots the time to solution breakeven point. The result show that, even though *Lazy Replication* reduces the shadows' execution rate, the expected time to solution is actually shorter than that provided by traditional replication. For example, when socket MTBF is 25 years traditional replication is viable at 97,600 sockets, whereas *Lazy Shadowing* is more efficient at just 52,700 sockets, representing a 46% improvement in expected time to solution.

Overhead	Method	# Sockets	# Main Sockets
60%	Checkpointing	100,000	100,000
60%	Replication	100,000	50,000
60%	Stretched	153,846	76,923
60%	Shadowing	124,998	62,499
80%	Checkpointing	100,000	100,000
80%	Replication	100,000	50,000
80%	Stretched	120,230	60,115
80%	Shadowing	110,636	55,318

Table 4: Available sockets for 20 MW budget and 200W per socket.

The improvement in time to solution is due to the fact that *Lazy Shadowing* can utilize additional sockets while consuming the same power, because shadows are consuming less power. This is illustrated in Table 4 which shows the active socket counts allowable given a 20 mega-watt fixed power budget. Both stretched and *Lazy Shadowing* have the ability to use additional sockets because they reduce the power consumed by the individual sockets. Stretched replication reduces the power consumed by all processes equally whereas *Lazy Shadowing* only reduces the power consumed by the replica sockets. This is the reason that the expected time to completion of *Lazy Shadowing* outperforms traditional replication. Stretched replication is able to add additional nodes but because it also reduces the processor speed of the main processes, the time to solution is higher than both traditional and *Lazy Shadowing*.

In pure replication the total amount of work remains constant but the number of sockets is half of that available to coordinated checkpointing. However, because with replication two sockets are used instead of one, the MTBF for the pair is greater than that provided in the single-socket case. The change in MTBF is what allows replication to outperform coordinated checkpointing at large scale. In *Lazy Shadowing*, instead of assuming half of the original sockets are replicas, we calculate the energy optimal σ_s. Then "add" as many additional sockets, while adhering to the power constraints. Stretched replication is similar to *Lazy Replication*, but both the shadow and main use less power.

In conclusion, the results show that *Lazy Shadowing* is both more energy efficient and produces solutions faster than traditional replication in power-limited systems. This is true for the majority of the exascale design space, illustrated by the region in the grey box in Figure 7.

3 Runtime Support for Jumping Shadow

The performance of Jumping 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:

- A reliable and efficient communication channel between the main process and its associated shadow
- A scalable and robust data structure to efficiently forward messages from mains to shadows, and
- Scalable and efficient algorithms to enable *shadow jumping*, with minimum disruption to computing progress.

In this project, 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 jumping functionalities of Shadows. The design of the communication layer will be fully interoperable with MPI, and its integration into the model will have no impact on MPI communication semantics or execution. Multiple challenges must be addressed, however, to enable an efficient and scalable RDMA-enabled Jumping Shadow. In the following, we explore potential research directions and discuss solutions for the design and implementation of this minimal communication layer. We will also explore the potential of an RDMA-enabled infrastructure to efficiently support the implementation of the jumping component of the Shadows.

3.1 Basic Message Forwarding Structure and Dynamics

In the Jumping Shadow resilience model, communications and message forwarding takes place between the main and its associated shadow. 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 Jumping Shadow, but also suffer performance impacting shortcomings. Careful consideration and analysis of the Jumping Shadow dynamics, however, reveal that the interaction between the main and its associated shadow involve both long messages, such content of parts of the main process's address space and large size data messages, and 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 application completes its execution. These dynamics open the door for the use of otherwise undesirable mechanisms, such as address registration and memory pinning. Furthermore, until failure occurs, the shadow has no interaction with other processes. Consequently, polling for communication from multiple processes is not required.

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.

3.2 Jumping Runtime Support and Execution

Jumping ensures that message buffers are flushed before the message queues overflow their capacity. It provides efficient means by which Jumping Shadow achieves forward progress, without execution, using a state that is consistent with that of the main process. Jumping 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 request jumping; (ii) an efficient mechanism to update the state of the shadow in an efficient and reliable manner that ensures consistency with the state of the main process.

The jumping 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 halting the virtual machine execution for a significant amount of time. Such an undertaking involves typically one or more of three phases, namely 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. The following observation, related to the characteristics and dynamics of the jumping process, highlight the difference between this process and live migration:

- In failure-induced jumping, 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.
- In forced jumping, the process can be achieved asynchronously, thereby affecting only related process.
- Depending on the nature of the application and the frequency of synchronization barriers, jumping can be achieved independently and in an uncoordinated fashion between groups of main-shadow pairs.
- 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 exascale applications, which typically involve tasks that are tightly coupled by data movement. Due to these differences, we will explore new scalable techniques to enable jumping 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 new techniques to:

- Update the CPU execution state of the shadow process,
- Incrementally copy the ”dirty” pages from the main address space to the shadow 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.
- Transfer the 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.
- Explore resource- and latency-aware, adaptive compression schemes to enhance the page transfer performance of the jumping 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.

3.3 Prototype Implementation

A prototype, *jsMPI*, will be built to experimentally evaluate the performance of Jumping Shadow. The prototype is aimed at running real-life scientific applications and understanding the performance impact of the different parameters in the design of Jumping Shadow. This effort will leverage, SrMPI, a software library developed by our research group, and augment its functionality to support the main components of Jumping Shadow, including shadow overloading, required consistency protocols, message-logging and message-forwarding protocols and state update operations in support of failure-induced and forced jumping. Currently, SrMPI supports replicating MPI ranks, which correspond to tasks in our model, and controlling speed in response to failures. The need to develop our own library stems from the lack of support of existing libraries for dynamic adjustment of speed execution and efficient support for Jumping Shadow’s consistency protocols [26, 43, 40]. SrMPI supports a parallel mode of *active* communication consistency, whereby a replica actively processes every request. Active replication requires additional system-wide messages, but makes error detection much easier. In this mode, the replicas only communicate with the other replica ranks and the main ranks only communicate with other main ranks.

The SrMPI library is implemented as a profiling library on top of OpenMPI. The implementation modifies all collective operations to use point-to-point communication primitives directly. This is achieved by implementing optimized versions of each collective operation supported in MPI. This approach provides the flexibility to implement replication solely in the point-to-point operations. Next, each point-to-point primitive function was modified to support communication with the replica ranks following the consistency protocols. Special care was taken to support MPI topologies, especially Cartesian topology used in MPI applications in the testing environment.

When SrMPI is enabled, the `MPI_Init()` function divides the available ranks into two different sets: main and shadow ranks. From the applications’ perspective, `MPI_COMM_WORLD` is cut in half and all associated functions such as `MPI_Comm_size()` and `MPI_Comm_rank()` are modified to return the proper rank and sizes. After the `MPI_Init()` function, the map between main and shadow ranks is fixed and available on each rank. By default, the main ranks are the lower half of the rank-space; however, this mapping can be overwritten using a configuration file or environment variables. Collective operations are written using the point-to-point communication functions; therefore, the implementation only needs to concern itself with the consistency of the point-to-point functions.

Enabling jsMPI: To support the semantics of Jumping Shadows, several design issues must be addressed.

- SrMPI only supports active replication. Passive replication has potential for a significant reduction in communication overheard. Two different approaches can be used to implement passive replication: either the main is responsible for pushing messages to the shadow, or the main can wait for receive requests from the replica. We will adopt the first approach, in the implementation of Jumping Shadow, and explore different approaches to guarantee the required consistency semantics. Special care must given to messages in transit, particularly when the main process fails before receiving the message.
- In order to switch the shadow of the failing main to recovery mode and trigger a failure-induced jumping, 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. 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.

- When implementing Jumping Shadow's functionalities into MPI, 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 jsMPI support mapping between the application request pointers and the multiple requests being maintained by SrMPI. To achieve this behavior, we will explore different approaches, focused the modification of the `MPI_Wait()` and `MPI_Test()` functions, 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.
- jsMPI assumes that there is a specified API for both determining available execution speeds and setting those speeds. Most systems have a specified set of execution "gears" which use a combination of both frequency and voltage scaling to achieve an execution speed. We will use the provided API to control execution rate of the shadow, both initially and upon the occurrence of a failure.
- We will incorporate the RDMA-enabled logical channel protocol into jsMPI, to achieve zero-copy efficient data transfer between the main process and its shadow. A concern concern is that messages must retain their message 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 shadow ranks. This allows the receive functions to guarantee order even if messages are interleaved between the main and shadow ranks.
- 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 Jumping 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.

The prototype is meant to be tested on a small-scale system, where all conditions can be kept under control. DoE has created three co-design centers: Exascale Co-Design Center for Materials in Extreme Environments (ExMatEx), at Los Alamos National Laboratory (LANL), Center for Exascale Simulation of Advanced Reactors (CESAR) at Argonne National Laboratory (ANL), and Center for Exascale Simulation of Combustion in Turbulence (ExaCT) at Sandia National Laboratory. Each center provides a collection of proxy applications that represent the parallel computing and communication patterns that real applications at extreme scale are expected to display. We will explore the set of applications and use them as benchmarks to understand the impact of shadow computing in the execution time of such programs on faulty hardware. In addition, we will explore the Mantevo Benchmark Suite [57]. This collection of codes represents the core algorithms in larger and more complex applications used at Sandia National Laboratory. Each mini-application is meant to represent the main characteristics of its corresponding science and engineering simulation. The main goal of mini-applications is to work as a lower entry point for people developing the vast ecosystem of HPC tools around traditional HPC applications. That includes compiler developers, network designers, architecture engineers, simulator programmers, language and programming model scientists, among others.

4 Related Work

Exascale presents some unique challenges to fault tolerance in exascale systems as faults are no longer an exceptional event [28, 55, 8, 11]. Rollback and recovery are predominate mechanisms to achieve fault tolerance in current HPC environments. In the most general form, the rollback and recovery mechanism involves the periodic saving of the current system state to stable storage, with the anticipation that in the case of a system failure, computation can be restarted from the most recently saved state prior to failure [21, 34, 60, 13, 34]. The identification of an error, before or during a checkpoint, requires that the application rollback to the previously completed checkpoint. Coordinated Checkpoint is a method to achieve a globally consistent state. Specifically, all processes coordinate with one another to produce individual states that satisfy the "happens before" communication relationship [14], which is proved to provide a consistent global state. Essentially, the algorithm provides a method for all processes involved to stop operation "at the same-time" and transfer their system state to a stable storage. 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 [23, 28, 55, 8, 11, 32, 49].

In uncoordinated checkpointing, processes record their states independently and postpone creating a globally consistent view until the recovery phase [62, 52]. The major advantage of this scheme is its potential for a reduced overhead during fault free operation [64]. However, the scheme requires that each process maintains multiple checkpoints and message logs, necessary to construct a consistent state during recovery. It can also suffer the well-known domino effect, which may result in the re-execution of the entire application [54]. One hybrid approach, known as communication induced checkpointing schemes, aims at reducing coordination overhead, [3, 9]. The approach, however, may cause processes to store useless states that are never used in future rollbacks. To address this shortcoming, "forced checkpoints" have been proposed [33, 36]. This approach may lead to unpredictable checkpointing rates. Although well-explored, uncoordinated checkpointing has not been widely adopted in HPC environments, due to its dependency on applications [65, 30].

One of the largest overheads in any checkpointing process is the time necessary to write the checkpoint to stable storage, which the application to pause its execution [49]. Incremental checkpointing attempts to address this shortcoming by only writing the changes since the previous checkpoint [15, 1]. This can be achieved using dirty-bit page flags [50, 51, 22]. Hash based incremental checkpointing, on the hand, makes use of hashes to detect changes [1, 15]. Another proposed scheme, known as copy-on-write, offloads the checkpointing process to a secondary task and only writes incremental checkpoints [42]. The main concern of these techniques is that some applications would require increase in their memory to support the simultaneous execution of the checkpoint and the application. It has been suggested that nodes in exascale systems should be configured with fast local storage, which improves the performance of checkpointing [2]. Multi-level checkpointing, which consists of writing checkpoints to multiple storage targets, can benefit from such a strategy [47, 58, 31]. This, however, may lead to increased failure rates of individual nodes and increased per-node cost [16]. Furthermore, it may complicate the checkpoint writing process and requires that the system track the current location of all process's checkpoints.

Process and state machine replication have long been used to provide fault tolerance in distributed [7, 37, 59] and mission critical systems [5]. To ensure consistent system state, all process messages must be delivered to all nodes running a given process, typically done by using a message ordering protocol [38]. Replication can be used to detect and correct system failures that are otherwise undetectable, such as silent data failures [48] and Byzantine faults [44, 12, 29, 63]. Replication was proposed as a viable alternative to checkpointing in HPC [28, 55, 8, 11, 41, 24, 10, 27]. Full and partial replication have also been proposed, to augment existing checkpointing techniques, and to guard against silent data corruption [28, 11, 48, 61, 20]. There are several different implementations of replication in the widely used MPI library, each with their different tradeoffs and overheads. The overhead can be negligible or up to 70% depending upon the communication patterns of the application [25]. Moreover, replication alone is not enough to guarantee fault tolerance since it is possible that all nodes executing a given process could fail simultaneously, thus replication is typically paired with some form of checkpointing. **To the best of our knowledge, Lazy Shadowing 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.**

5 Research Tasks, Milestones and Management Plan

The activities of this project will be carried out over a three-year period. The research agenda includes three major thrusts: The first thrust is focused on developing essential algorithms and components of "Lazy Shadowing" and studying its performance and behavior, in different environments and for different application requirements. The second thrust is focused on developing and assessing the performance the minimal RDMA-enabled channel, failure-induced and forced jumping, and rate execution contol components of sjMPI. The third thrust focuses on fully integrating Jumping Shadow in MIP and evaluating its resiliency. These thrusts are organized in the following set of tasks:

- Task 1: Develop a power-aware optimization framework to model and study energy and power consumption of Jumping Shadow. This framework will be used to determine a shadow's optimlal execution rates to minimize energy, meet power constraints and achieve the expected time-to-solution.
- Task 2: Develop power-aware, fault-isolated mapping schemes which use colocating to reduce execution rate.
- Task 3: Develop a simulation framework to validate the analytical model and assess the performance of the different heuristics in large-scale environments.
- Task 4: Integrate the rate execution control and shadow maping schemes in sjMPI.
- Task 5: Develop 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 and failure models.

- Task 6: Develop the send() and receive() primitives, using InfiniBand RDMA mechanisms, to support efficient communications between a process and its shadow.
- Task 7: Modify relevant MPI functions to support sjMPI funcationalities, including:
    1. The send and the send/receive primitives, which instead of blocking, return a request pointer to the application, used by `MPI_Wait()`, `MPI_Wait()`, `MPI_Test()` and `MPI_Finalize()`;
    2. The receive primitives, which allow posting a receive request for the main or shadow rank depending on the node type from which the function is called;
    3. The test() and wait() primitives, which return a modified request pointer provided by the non-blocking send and receive methods, and
    4. Toplogy functions, specifically when creating a Cartesian topology, which requires that the ranks must be ordered consistently to ensure that any application using these methods received consistent results.
- Task 8: Carry out a thorough analysis of sjMPI resiliency for different applications and failure models.

The PIs form a strong research team. 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. Drs. Melhem and Znati recently collaborated on a research project, funded by NSF, which led to the design and implementation of SrMPI. Dr. Kant's expertise resides in the emerging field of security and robustness in cloud computing, secure configuration management for large IT systems, and energy efficiency and sustainability in computing and communications. Dr. Meneses's expertise includes building effective computer science infrastructure to accelerate scientific discovery. His research work focuses on developing models, algorithms, and tools to aid in solving prominent problems in computational science, in current and future heterogeneous supercomputer environments.

Dr. Melhem will be the project director and responsible for the management of the project. He will lead the development of a power-aware framework for enegery consumption and control. Efforts related to Task 2 will be led by Dr. Znati and Drs. Kant and Znati will co-lead the development and simulation framework related to Task 3. Drs. Znati and Meneses will lead the integration of shadow execution control into sjMPI. Drs. Kant and Meneses will co-lead the development of an experimental study to evaluate and compare power and energy savings of Jumping Shadow, in different settings. Drs. Meneses and Znati will lead the research efforts related to Taks 7. Drs. Melhem and Meneses will co-lead the analysis of sjMPI resiliency. All members of the research team will actively particiapte in all aspects of the project, including the development and evaluation of the communications protocols, the runtime support algorithms and data structure, and their integration and evaluation of into sjMPI. The planned schedule of tasks and milestones, along with main roles and responsibilities of the PIs, are shown the following table:

	Q1	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Q11**	**Q12**	**Lead PIs**	**Task Budget**
Task 1	🟦	🟦	🟦	🟦	🟦								Melhem	$153,000
Task 2	🟩	🟩	🟩										Znati	$100,000
Task 3		🟨	🟨	🟨	🟨	🟨							Kant & Znati	$165,000
Task 4		⬛	⬛	⬛	⬛	⬛	⬛						Znati & Meneses	$150,000
Task 5					🟥	🟥	🟥	🟥	🟥	🟥			Kant & Meneses	$180,000
Task 6							🟩	🟩	🟩	🟩	🟩		Znati & Melhem	$175,000
Task 7							🟦	🟦	🟦	🟦			Meneses & Znati	$143,000
Task 8									🟪	🟪	🟪	🟪	Melhem & Meneses	$170,000
Task shedule and management plan														$1,236,000

Regular meetings will be held by the PIs, to evaluate current plans and assess research progress. A yearly one-day workshop will be held to discuss current state of the research effort, prioritize tasks in the project and discuss future plans and task assignment. We will maintain a shared *dropbox* and a project web site to facilitate sharing of documents, plans, and test results. The website will describe the main objectives of the project and report on its findings and research outcome. We will maintain a code repository for the development and experimentation efforts. We will seek collaboration with other DoE supported projects, and will leverage the DoE funded software, currently available in the design and implementation of several components.

APPENDIX 1: Bibliographical Sketches

Rami Melhem

PROFESSIONAL PREPARATION:

University of Pittsburgh	Computer Science	Ph.D. - 1983
University of Pittsburgh	Computer Science	M.S. - 1981
University of Pittsburgh	Mathematics	M.A. - 1981
Ein-Shams University, Egypt	Mathematics	B.S. - 1978
Cairo University, Egypt	Electrical Engineering	B.S. - 1976

APPOINTMENTS:

1984	*Research Associate*, University of Pittsburgh (January - September)
1984-1987	*Assistant Professor* of Computer Science, Purdue University,
1985-1986	*Visiting Assistant Professor* of Mathematics, Univ. of Pittsburgh
1986-	*Assistant/Associate/Full Prof*. of Computer Science, Univ. of Pittsburgh
1993-	*Associate Prof. and Prof*. of Electrical Eng., Univ. of Pittsburgh
2000- 2009	*Chair* of the Computer Science Department, Univ. of Pittsburgh.

CURRENT RESEARCH INTERESTS:
Power Aware Computing, Fault-tolerant and Real-time Systems, High Performance Computer Architecture, Parallel and Distributed Computing and Optical interconnection networks.

HONORS
Fellow of the IEEE

SELECTED PUBLICATIONS:

M. Moeng, R. Melhem and A. Jones, "Weighted-Tuple Synchronization for Parallel Architecture Simulators", *Proc. Int. Symp. On Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2014)*, Paris, France.

X. Cui, B. Mills, T. Znati and R. Melhem, "Profit Maximization for Resilient Cloud Computing", *Proc. of the International Conference on Cloud Computing and Services Science (CLOSER 2014)*, Barcelona, Spain.

M. Zhou, Y. Du, B. Childers, R. Melhem, D. Mosse, "Writeback-Aware Bandwidth Partitioning for Multi-core Systems with PCM", *Proc. of the International Conference on Parallel Architectures and Compilation Techniques (PACT 2013)*, Edinburgh, Scotland.

Y. Du, M. Zhou, B. Childers, D. Mosse and R. Melhem, "Bit Mapping for Balanced PCM programming", *Proc. of the International Symposium on Computer Architecture (ISCA 2013)*, Tel Aviv, Israel.

Y. Li, R. Melhem and A. Jones, "Practically Private: Enabling High Performance CMPs Through Compiler-assisted Data Classification", *Proc. of the Int. Conference on Parallel Architectures and Compilation Techniques (PACT 2012)*, Minneapolis, MN.

R. Melhem, R. Maddah and S. Cho, "RDIS: A Recursively Defined Invertible Set Scheme to Tolerate Multiple Stuck-At Faults in Resistive Memory", *Proc. of the 42nd IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN 2012)*, Boston, MA.

M. Zhou, R. Melhem, B. Childers and D. Mosse, "Writeback-aware Partitioning and Replacement for Last-Level Caches in Phase Change Main Memory Systems", *ACM Transactions on Architecture and Code Optimization*, vol 8, issue 4, article 53 (2012).

S. Bock, B. Childers, R. Melhem, D. Mosse, Y. Zhang, "The Impact of Useless Write-Backs on the Endurance and Energy Consumption of PCM Main Memory", *Proc. of the IEEE Int. Symp. on Performance Analysis of Systems and Software (ISPASS 2011).*

Y. Xu, J. Yang and R. Melhem, "A Process Variation Tolerant Design for Nanophotonic Networks," *Proc. of the International Symposium on Computer Architecture (ISCA 2012).*

A. Abousamra, A. Jones and R. Melhem, "Co-Design of NoC and Cache Organization for Reducing Access Latency in Chip Multiprocessors", *The IEEE Transactions on Parallel and Distributed Systems*. vol. 23 no. 6, pp. 1038-1046 (2012).

PROFESSIONAL AND SYNERGISTIC ACTIVITIES :

Editor: IEEE Trans. on Computers (1991 – 96, 2011 -)The International Journal of Embedded Systems (2004 -), Journal of Parallel and Distributed Computing (2003 - 10), Computer Architecture Letters (2001 - 2010), IEEE Trans. on Parallel and Distributed Systems (1998 - 2002), Springer Book Series in Computer Science (1997 - 09).

Program Committee Member in the last two years: Hot Interconnects (2013,14), The IEEE Conference on Green Computing GCC (2013), IEEE Int. Conf. on High Performance Computing HiPC (2014), The Int. Parallel and Distributed Processing Symposium IPDPS (2013,14), The Int. Conf. on Supercomputing (2013), The Int. Conf. on Parallel Arch. and Compilation Techniques, PACT (2014), The IEEE Int. Conf. on Big Data and Cloud Computing, BDCloud(2014), The IEEE Int. Conf. on Parallel and Distributed Systems, ICPADS(2014), The Int. Green Computing Conf., IGCC(2013, 14).

Established the Technology Leadership Institute at the University of Pittsburgh. This is a summer program for high school minority students interested in pursuing a career in information technology.

Established an Industry Advisory Board in the Department of Computer Science at the University of Pittsburgh.

COLLABORATORS:

E. Elnozahi (Kaust), Eugen Schenfeld (IBM), R. Gupta (U. of Arizona), R. Graybill (ISI), R. Libeskind-Hadas (Harvey-Mudd College), R. Roskies (Pittsburgh Supercomputing Center), H. Shafi (Microsoft Research), Dale Rickard (BAE Systems), Jeff Robertson (BAE Systems), Steve Crago (ISI), Mohammed Amduka (Lockheed Martin), Iyes Robert (INRIA), Krishna Jha (Lockheed Martin), Mazin Yousef (IBM).

PH.D. ADVISOR AND ADVISEES:

Werner Rheinboldt (advisor), Sultan Alam, Zincheng Guo, Chunming Qiao, Chun Gong, John Ramirez, Sunondo Ghosh, Nimish Shrivatava, Xin Yuan, Charles Salisbury, Sylvain Lauzac, Libin Dong, Hakan Aydin, Dakai Zhu, Cosmin Rusu, Nevine AbouGhazaleh, Sameh Gobriel, Sherif Khattab, Ruibin Xu, Mohamed Hammoud , Mahmoud Elhaddad , Shuyi Shao, Ahmed Abousamra, Michel Hanna.

Taieb F. Znati, Biography Sketch

Department of Computer Science
University of Pittsburgh
(412) 624 8217
Znati@cs.pitt.edu

EDUCATION

1984-1988	Michigan State University, East Lansing, Michigan. Ph.D. Degree in Computer Science, 1987.
1981-1983	Purdue University, West Lafayette, Indiana. Master of Science degree in Computer Science, 1983
1976-1981	Institue Nationale de Recherche Informatique et Automatique, Paris, France. Ingénieur Principal D'Informatique Faculté Des Sciences, Ecole D'informatique, Tunis. Diplôme D'Etudes Supérieurs en Mathématiques et Physiques.

PROFESSIONAL EXPERIENCE

2007- 2010	National Science Fondations, CISE/CNS. Director, Division of Computer and Network Systems.
2000- 2004	National Science Foundations, CISE/ANIR and CISE/CNS. Senior Program Director, Division of Computer and Network Systems. Information Technology Research Initiative 03 – Coordination Committee Chair
1988-	University of Pittsburgh, Pittsburgh, Pennsylvania. Computer Science Department and Information Science department. Professor of Computer Science and Telecommunications.
1984-1988	Michigan State University, East Lansing, Michigan. System Manager and Network Coordinator Research and Teaching Assistant.
1981-1983	Purdue University, Computer Science Department Computer Science Teaching Assistant.

RELATED PUBLICATIONS

1. XiaolongCui, Bryan Mills, Taieb Znati * and Rami Melhem, "Shadow Replication: An Energy-Aware, Fault-Tolerant Computational Model for Green Cloud Computing", *Energies* **2014**, *7*(8), 5151-5176;
2. Bryan Mills and Taieb Znati and Rami Melhem and 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
3. Bryan Mills and Taieb Znati and Rami Melhem, "Shadow Computing: An Energy-Aware Fault Tolerant Computing Model", In Proceedings of the International Conference on Computing, Networking and Communications (ICNC),2014.
4. T. N. Dinh, Y. Xuan, M. T. Thai, and P. M. Pardalos, " On New Approaches of Assessing Network Vulnerability: Hardness and Approximation", IEEE/ ACM Transactions on Networking (ToN), 2011.
5. Hui Ling, Taieb Znati, and Louise Comfort,"Design Resilient Systems: Integrating Science, Technology and Policy in International for Multiple Query Processing in Wireless Sensor Networks", Risk Reduction", in "Designing Resilience: Preparing for Extreme Events", Chap. 13, May 2010

OTHER RELEVANT PUBLICATIONS

1. Thang N. Dinh, Ying Xuan, My T. Thai, E. K. Park, and Taieb Znati, "On Approximation of New Optimization Methods for Assessing Network Vulnerability", INFOCOM 2010, pp. 2678-2686

2. Sangpachatanaruk, Chatree; Znati, Taieb, “Prototyping and Analysis of an Ontology-based Personalized Web Service Architecture”, IEEE ANSS Proceedings, 7-14 2008.
3. Mills, Bryan N.; Znati, Taieb, “Increasing DHT data security by scattering data”, ICCCN, 430-434 2008.
4. Yang Zhao, Zhiguang Qin, Youtao Zhang, and Taieb Znati, ”SDC: Secure Data Collection for Time Based Queries in Tiered Wireless Sensor Networks”, RTCSA 2009, pp. 255-262.
5. My T. Thai, Ying Xuan, Incheol Shin, Taieb Znati: On Detection of Malicious Users Using Group Testing Techniques. ICDCS 2008: 206-213.

SYNERGISTIC ACTIVITIES

1. Director, Division of Computer Systems and Networks; Senior Program Director in Advanced Network Infrastructure and Research; and Coordination Committee Chair of the ITR Initiative – managed and supervised programs in research and education and activities in broadening participation in computing, including the Broader Impacts for Research and Discovery Summit.
2. Member of the Original Telecommunications Program Committee, joint program between Information Science Department, Computer Science Department, School of Electrical Engineering and School of Business: Conceived and developed curricular material and pedagogical methods for the networking and communications core area of the program. Developed teaching and training tools for network protocol animation.
3. Original Designer of Grid@Pitt – A cluster-based architecture for teaching and research in parallel and data-intensive computing.
4. Network Focused College– Member of the original team responsible for the design and development of curricular material, teaching and training tools for AT&T and DEC training and education program.
5. **General Chair**: IEEE GLOBECOM 2010, IEEE INFOCOM 2005, International Conference on Ubiquitous and Pervasive Healthcare (UbiCare 2007); First IEEE Conference on Sensor and Ad-Hoc Communications Networks (SECON 2005); **General Chair and Executive Committee Member**: Wireless Algorithms, Systems and Applications (WASA 11). **Associate Editor**: Wireless Networks; Journal of Ad-Hoc Networks; IEEE Transactions on Parallel and Distributed Systems; and Program Chair: International Conference on Bio-Inspired Computing (BIC’11)

COLLABORATORS & OTHER AFFILIATIONS

1. **Research Collaborators:** Hui Zhang, Computer Science Department, Carnegie Melon University; Xiuzhen Cheng, Computer Science Department, George Washington University; Randy Choi, Computer Science Department, University of Florida at Gainesville; Kazem Sohraby, Electrical Engineering, University of Arkansas; Dan Minoli, AT&T Research; Mohsen Guizani, Western Michigan University; Bechir Hamdaoui, Oregon State University; My Thai, University of Florida at Gainesville; Weili Wu, University of Texas at Dallas.
2. **Graduate Advisors:** Dr. Lionel Ni, Michigan State University and University of Hong Kong (Ph.D. Advisor), and Dr. Herbert Schwetman: Purdue University and … (MS Advisor).
3. **Thesis Advisor and Postgraduate-Scholar Sponsor**: Brian Field (Comcast Cable); Tawfig Al Rabiah (Industrial Property Authority, KSA); Abdelrahman Al Jadhai (Riadh University, KSA); Robert Simon (George Mason University); Michael Boykin (Sara Lee, OH); Bruce McDonald (North Eastern); Gretchen Lynn (Wesleyan College); Tareen Saadullah (QualCom); Massaru Okuda (Murray State University); Chatree Sangpachatanaruk (NetApp); Subrata Acharya (Towson Univ.); Anandha Gopalan (Imperial College London); Hui Ling (ECI Telecom); Ihasan Qazi (Swinburne University, Melbourne).

Esteban Meneses

Professional Preparation

PhD in Computer Science
University of Illinois at Urbana-Champaign
August, 2013

Master of Science in Bioinformatics
University of Illinois at Urbana-Champaign
May, 2013

Master of Science in Computer Science, Magna Cum Laude
Costa Rica Institute of Technology
March, 2007

Bachelor of Engineering in Computing Engineering, with Honors
Costa Rica Institute of Technology
September, 2001

Appointments

Research Assistant Professor — **University of Pittsburgh**
September, 2013 - present — Pittsburgh, Pennsylvania

Research Assistant — **University of Illinois at Urbana-Champaign**
September, 2007 - August, 2013 — Urbana, Illinois

Research Aide — **Argonne National Laboratory**
June, 2010 - August, 2010 — Chicago, Illinois

Adjunct Professor — **Costa Rica Institute of Technology**
July, 2004 - August, 2007 — Cartago, Costa Rica

Research and Development Programmer — **PrediSoft**
January, 2004 - June, 2004 — San José, Costa Rica

Research and Development Programmer — **ArtinSoft Corp**
January, 2001 - November, 2003 — San José, Costa Rica

Products

1. Esteban Meneses, Xiang Ni, Gengbin Zheng, Celso L. Mendes and Laxmikant V. Kalé. **Using Migratable Objects to Enhance Fault Tolerance Schemes in Supercomputers**. *IEEE Transactions on Parallel and Distributed Systems (TPDS). In press.*
2. Esteban Meneses, Osman Sarood and Laxmikant V. Kalé. **Energy Profile of Rollback-Recovery Strategies in High Performance Computing**. *Parallel Computing (ParCo)* 40, number 9, 2014, pp 536-547.
3. Osman Sarood, Esteban Meneses and Laxmikant Kalé. **A Cool Way of Improving the Reliability of HPC Machines**. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC) 2013.* Denver, Colorado, USA. November, 2013. **Acceptance rate: 20%**.
4. Xiang Ni, Esteban Meneses, Nikhil Jain and Laxmikant Kalé. **ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection**. *International Conference for High Performance Computing, Networking, Storage and Analysis (SC) 2013.* Denver, Colorado, USA. November, 2013. **Acceptance rate: 20%**.
5. Emmanuel Jeannot, Esteban Meneses, Guillaume Mercier, François Tessier and Gengbin Zheng. **Communication and Topology-aware Load Balancing in Charm++ with TreeMatch**. *IEEE International Conference on Cluster Computing (Cluster) 2013.* Indianapolis, Indiana, USA. September, 2013. **Acceptance rate: 31%**.
6. Esteban Meneses, Osman Sarood and Laxmikant V. Kalé. **Assessing Energy Efficiency of Fault Tolerance Protocols for HPC Systems**. *24th International Symposium on Computer*

Architecture and High Performance Computing (SBAC-PAD). New York City, New York, USA. October, 2012. **Acceptance rate: 28.12%**. **Julio Salek Aude Best Paper Award**.

7. Xiang Ni, Esteban Meneses and Laxmikant V. Kalé. **Hiding Checkpoint Overhead in HPC Applications with a Semi-Blocking Algorithm**. *IEEE International Conference on Cluster Computing (Cluster) 2012*. Beijing, China. September, 2012. **Acceptance rate: 28.86%**.
8. Esteban Meneses, Greg Bronevetsky and Laxmikant V. Kalé. **Dynamic Load Balance for Optimized Message Logging in Fault Tolerant HPC Applications**. *IEEE International Conference on Cluster Computing (Cluster) 2011*. Austin, Texas, USA. September, 2011. **Acceptance rate: 27.85%**.
9. Thomas Ropars, Amina Guermouche, Bora Uar, Esteban Meneses, Laxmikant V. Kalé and Franck Cappello. **On the Use of Cluster-Based Partial Message Logging to Improve Fault Tolerance for MPI HPC Applications**. *European Conference on Parallel Processing (Euro-Par) 2011*. Bordeaux, France. September, 2011. **Acceptance rate: 29.9%**.
10. Gengbin Zheng, Abhinav Bhatele, Esteban Meneses and Laxmikant V. Kalé. **Periodic Hierarchical Load Balancing for Large Supercomputers**. *International Journal for High Performance Computing Applications (IJHPCA)*, 2011.

Synergistic Activities

- ◇ **University Service**. XSEDE Campus Champion at the University of Pittsburgh since 2013.
- ◇ **Community Service**. Program Committee Member and Reviewer: International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2014), International Conference of the Chilean Society of Computer Science (SCCC 2014), IEEE Transactions on Parallel and Distributed Systems (TPDS), Springer's Cluster Computing (CLUS), ACM Transactions on Architecture and Code Optimization (TACO).
- ◇ **Student Service**. Member of the *CS Graduate Student Academic Council*. Department of Computer Science, University of Illinois at Urbana-Champaign, Fall, 2009 - Spring, 2011.
- ◇ **Mentoring**. Sponsor of the ACM Student Chapter at the Costa Rica Institute of Technology from August, 2005 to July, 2007.
- ◇ **Teaching**. Publicly available lecture notes and related material for the class *Introduction to High Performance Computing Systems* (CS-1645) at the University of Pittsburgh.

Collaborators & Other Affiliations

- **Collaborators and Co-Editors**: Laxmikant V. Kalé (UIUC), Terry Jones (ORNL), Celso L. Mendes (NCSA), Gengbin Zheng (NCSA), Greg Bronevetsky (LLNL), Franck Cappello (ANL), Abhinav Bhatele (LLNL), Amina Guermouche (UVSQ), Nikhil Jain (UIUC), Emmanuel Jeannot (INRIA), Guillaume Mercier (INRIA), Xiang Ni (UIUC), Thomas Ropars (EPFL), Francois Tessier (INRIA), Bora Uçar (INRIA).
- **Graduate Advisors and Postdoctoral Sponsors**: Prof. Laxmikant V. Kalé (University of Illinois at Urbana-Champaign).

Krishna Kant

Office phone: (215) 204-8450
Email: kkant@temple.edu **Home page** : http://www.kkant.net

Education:

University of Texas at Dallas, Richardson, TX	Computer Science	Ph.D. (1981).
Indian Institute of Science, Bangalore, India	Electrical Comm. Eng	M.S. (1977).
Indian Institute of Technology, Kanpur, India	Electrical Engineering	B.Tech (1975).

Appointments

Nov 2013 – Present: Professor, Computer and Information Sciences (CIS) department, Temple University, Philadelphia, PA

2010 – Nov 2013: Research professor, Center for Secure Information Systems (CSIS), George Mason University. Also served as a Program director in the Computer and Networks division of CISE directorate at the National Science Foundation until Sept 2013.

2008 - 2010: Program director, in the Computer and Networks division of CISE directorate at the National Science Foundation

1997 - 2008: Senior Performance Analysis Engineer, Digital Enterprise Group, Intel Corp, Hillsboro, OR. Focused on power management of platform resources, asset localization in data centers, network virtualization, high speed networking, and hardware protocol acceleration.

1991-1997: Senior Member of Technical Staff; Engineering, Performance, and Control department, Telcordia (formerly Bellcore), Red Bank, NJ. Areas of work included automated service provisioning, congestion control and link error monitoring in both narrow-band and broadband SS7 (signaling system No. 7) networks, and engineering issues relating to SS7 and intelligent services running atop SS7.

1985-1990: Assistant (1985-1989) and then tenured associate professor of Computer Science at Pennsylvania State University, University Park, PA. Research and teaching in operating systems, database systems, computer system performance modeling, distributed systems, and fault-tolerant computing. Published a highly regarded graduate textbook on mathematical performance modeling, and supported the first networked laboratory of SUN workstations.

1981-1984: Assistant Professor of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL. Research and teaching in fault-tolerant computing, operating systems, and software engineering.

Five Related Products:

1. J. Gim, Y. Won, T. Hwang, and K. Kant, "SmartCon: Smart Context Switching for Fast Block Devices", to appear in ACM transactions on Storage.
2. K. Kant, “ Multistate Power Management of Communication Links ”, Proc. of COMSNET 2010, Bangalore, India, Jan 2011
3. K. Kant, M. Murugan, D. Du, “Enhancing Data Center Sustainability Through Energy Adaptive Computing”, ACM Journal of Emerging Technologies in Computing Systems, Vol 8, No 4, Oct 2012.
4. K. Kant, " Data Center Evolution: A Tutorial on State of the Art, Issues, and Challenges ", Elsevier Computer Networks Journal, Dec 2009.

5. M. Murugan, K. Kant, D. Du, and A. Raghavan, "FlexStore: A flexible and Adaptive Storage Framework for Deduplicated Virtual Disks", Proc. of MASCOTS, Paris, France, Sept 2014.

Five Other Products:

1. K. Kant, " A Control Scheme for Batching DRAM Requests to Improve Power Efficiency ", Proc. of ACM Sigmetrics 2011, San Jose, CA, June 2011.
2. K. Kant and C. Deccio, "Security and Robustness in the Internet Infrastructure", in Handbook on securing cyber-physical systems, Morgan Kaufman, S. Das, K. Kant, N. Zhang (eds.), Feb 2012.
3. M. Murugan, David H.C. Du and K. Kant, "On the Interconnect Energy Efficiency of High End Computing Systems", Sustainable Computing: Informatics and Systems, Elsevier Press, April 2012.
4. M. Le, K. Kant and S. Jajodia, "Consistency and Enforcement of Access Rules in Cooperative Data Sharing Environment ", Elsevier Computers and Security (COSE) Journal, Vol 41 (March 2014), pp 3-18.
5. C. Deccio, J. Sedayao, K. Kant, and P. Mohapatra, " A Case for Comprehensive DNSSEC Monitoring and Analysis Tools ", Proc. of SATIN (Securing and Trusting Internet Names) 2011, Teddington, UK, April 2011

Synergistic Activities:

Dr. Kant carries a wide range of experience in academia, industry and government in a variety of computer science fields. His current areas of research interest include: Energy Efficiency and Sustainability issues in computing & communications, Security and robustness issues in cloud computing and name resolution, Configuration management issues in data centers and other large systems, networking technologies and systems. He also has expertise in a number of other areas including: Traffic characterization and congestion control, Low-level architectural modeling, Acceleration technologies (security, XML, TCP, etc.), Peer to Peer computing, Mathematical performance modeling, Telecommunication systems and Internet technologies, data center design issues.

At NSF, Dr. Kant drove the NSF wide sustainability initiative that includes programs on various facets including management of hazards and disasters and computing for sustainability. Dr. Kant has been involved with the development of a number of tools including DNSviz for DNS troubleshooting, LMPOWER for detailed low-level modeling of power management activities of memory subsystems, Geist for generation of web traffic with sophisticated traffic characteristics, DCLUE for modeling database systems, etc. Dr. Kant has also given comprehensive tutorials in major conferences on a variety of topics including peer to peer computing, large scale Internet failures, and power/thermal management of data centers.

He has also published a graduate textbook on computer systems performance modeling (McGraw Hill 1992), and edited a number of books, most recently *Handbook on Securing Cyber-Physical Infrastructures: Foundations and Challenges*, Morgan Kaufman, Feb 2012. He also serves on editorial board of several journals.

Collaborators and Co-Editors: Sajal Das (MST), David Du (U/Minn), Sushil Jajodia (GMU), Scott Midkiff (Virginia Tech), Prasant Mohapatra (UC/Davis), Lei Wang (U/Conn), Nan Zhang (GWU),

Graduate Advisor: Dr. Abraham Silberschatz (Yale)

Thesis Advising: Amit Sahoo (UC/Davis, now Cisco), Lihua Yuan (UC/Davis, now Microsoft), Casey Deccio (UC/David, Now Sandia Labs), Neha Udar (SIU, now with Intel), Muthukumar Murugan (U/Minn, now HP), Meixing Le (GMU, now Cisco), Mayank Raj (Missouri Univ. of S&T).

APPENDIX 2: Current and pending support

Melhem, Rami.

Active grants

NSF: CNS1252306 ($299,800) 01/2013 – 12/2014 support for 0.75 month per year
CSR: EAGER: *Exploratory Research on Scalable Resiliency Through Shadow Computing and Differential Data Replication*
The major goal of this project is to formulate the concept of shadow computing in a cloud environment where processes are independent (no communication) and synchronization occurs at the end of a long computation phase.

NSF: CCF-1064976 ($809,600) 03/2011 – 2/2015 support for 1.00 month per year
PI: Alex Jones
SHF: Medium: *Compiler and Chip Processor Co-Design for Scalable Efficient Data Access and Communication*
The major goal of this project is to study cross layer optimization of CMP cache and network-on-Chip performance and power consumption.

GRO Program ($124,331) 02/2014 – 01/2015 support for 1.00 month
Constructing Scalable, Energy Efficient and Reliable Main Memory with STT-RAM
The major goal of this project is to explore the application of the new Spintronic technology to large memory and investigate solutions to mitigate the problems related to excessive power consumption and low reliability resulting from the poor write performance of this emerging memory technology.

pending grants

Quatar Research Foundation ($269,395) 10/2014 – 7/2017 support for 1 month
PI: M. Hammoud
Scalable Analytics Engine for Big Graphs on the Cloud

Overlap

The NSF grant (CNS1252306) is an Eager grant specifically awarded to perform the preliminary work needed to prepare a proposal for a larger project. The preliminary results reported in the current proposal (submitted to DoE) were performed as part of this Eager project.

Znati, Taieb.

Active grants

NSF: CNS1252306 ($299,800) 01/2013 – 12/2014 support for 0.75 month per year
PI: Rami Melhem
CSR: EAGER: *Exploratory Research on Scalable Resiliency Through Shadow Computing and Differential Data Replication*
The major goal of this project is to formulate the concept of shadow computing in a cloud environment where processes are independent (no communication) and synchronization occurs at the end of a long computation phase.

NSF: CNS-1162159 ($331,735) 07/2012 – 06/2016 support for 1 month per year
NeTS: Medium: *Collaborative Research: Integrated Dynamic Spectrum Access for Throughput, Delay, and Fairness Enhancement*
The central theme of this project is to investigate methodologies and theories to enhance throughput, delay, and fairness of cognitive radio networks via integrated dynamic spectrum access. The focus is on dynamic spectrum access is of high national interest and can create significant impact on spectral usage policies and related industries in the telecommunication and information technology sectors.

NSF: OCE-1331463($3,000,000) 09/2013 – 08/2017 support for 1 month per year
PI: Louise Comfort
Hazards SEES Type 2: From Sensors to Tweeters: A Sustainable Sociotechnical Approach for Detecting, Mitigating, and Building Resilience to Hazards
This project addresses the challenge of building resilience to hazards. The goal is to define, design, and demonstrate an interdisciplinary, dynamic process that will transform societal understanding of risk and enable self-organized, collective action to support the resilient management of hazards. This study will identify and model the interactions among physical, engineered, and sociotechnical systems that occur in hazard emergence and response as a complex, adaptive system of systems (CASoS) to enhance resiliency in practice and enable communities to manage the risk of hazards within existing resource and time constraints.

NSF: CNS12 53218 ($187,950) 09/2012 – 08/2015 support for 0.5 month per year
PI: Kirk Pruhs
A Framework for joint optimization of power management and performance in virtualized, heterogeneous cloud computing environments
The main objective of the proposed research project is to develop holistic approaches to the design of power optimization strategies for heterogeneous cloud computing environment. The focus will be on exploring the tradeoffs among the cloud computing physical infrastructure, including power-managed servers and cooling technology, virtualized infrastructure, and the middleware and service infrastructure.

No pending grants

Overlap

The NSF grant (CNS1252306) is an Eager grant specifically awarded to perform the preliminary work needed to prepare a proposal for a larger project. The preliminary results reported in the current proposal (submitted to DoE) were performed as part of this Eager project.

Esteban Meneses-Rojas

<u>No active or pending grants</u>

Krishna Kant

Active grants

NSF, CNS-1407876 ($299,353),   05/01/2014 -- 04/30/2016, Support for 1.0 mo
CoPI: None
CSR: EAGER: Quality of Configuration in Large Scale Data Centers
The goal of this grant is to characterize quality of configuration of large scale systems that can be used to detect misconfigurations and enhance configuration robustness

NSF, CNS-1422921 ($249,997), **07/**01/2014 – 06/30/2017, Support for 0.5 mo
CoPI: None
CSR:Small: Collaborative Research: Software Defined Energy Adaptation in Large Scale Data Centers
The goal of this research is to define and build software-defined architecture for comprehensive energy management and adaptation in a data center.

NSF, IIP-1439672 ($325,000), 09/01/2014 – 08/31/2019, support for 0.25 mo
CoPI; Jie Wu
Collaborative Research: A Multi-University I/UCRC Phase II Center on Intelligent Storage
This grant supports the establishment and running of a Temple University node of an IUCRC (Industry University Cooperative Research Center) focused on data center energy management issues.

pending grants

NSF, CNS-1461932 ($299,999), 05/01/2015 – 04/30/2018, support for 0.25 mo
CoPI: Jie Wu, Solbodan Vucetic
BDD: Dynamic Evolution of Smart-Phone Based Emergency Communications Network
The purpose of this research is to research flexible communications protocols for emergency networks and support data analytics driven dynamic network reconfigurations.

NSF, CNS-1505844, ($236,340), 06/01/2015 – 05/30/2018, support for 0.1 mo
Co-PI: Avinash Srinivasan
CPS:Breakthrough: Collaborative Research: Securing Smart Grid by Understanding Communications Infrastructure Dependencies
The goal of this project is examine the interaction between the power grid and the supporting communications infrastructure and study its implications for security and robustness of the power grid.

APPENDIX 3: Bibliography and References cited

[1] 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.

[2] Sean Ahern, Arie Shoshani, Kwan-Liu Ma, Alok Choudhary, Terence Critchlow, Scott Klasky, Valerio Pascucci, Jim Ahrens, E. Wes Bethel, Hank Childs, Jian Huang, Ken Joy, Quincey Koziol, Gerald Lofstead, Jeremy S Meredith, Kenneth Moreland, George Ostrouchov, Michael Papka, Venkatram Vishwanath, Matthew Wolf, Nicholas Wright, and Kensheng Wu. Scientific discovery at the exascale, a report from the doe ascr 2011 workshop on exascale data management, analysis, and visualization, 2011.

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

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

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

[6] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, Katherine Yelick, and Peter Kogge. Exascale computing study: Technology challenges in achieving exascale systems, 2008.

[7] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. *SIGOPS Oper. Syst. Rev.*, 21(5):123–138, November 1987.

[8] 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.

[9] 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.

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

[11] 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.

[12] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

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

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] M. el Mehdi Diouri, O. Gluck, L. Lefevre, and F. Cappello. Energy considerations in checkpointing and fault tolerance protocols. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, 2012.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 78:1–78:12, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[30] 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.

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

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

[33] 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.

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

[35] Peter M. Kogge and David R. Resnick. Yearly update: Exascale projections for 2013. Technical Report SAND2013-9229, Sandia National Laboratories, Albuquerque, New Mexico, 2013.

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

[37] Leslie Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Trans. Program. Lang. Syst.*, 6(2):254–280, April 1984.

[38] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[39] Lawrence Livermore National Laboratory. ASC sequoia benchmark codes. `https://asc.llnl.gov/sequoia/benchmarks/`, 2013.

[40] 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.

[41] 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.

[42] 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.

[43] 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 23rd 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.

[44] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM J. Res. Dev.*, 6(2):200–209, April 1962.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

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

[54] 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.

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

[56] Rolf Riesen, Kurt Ferreira, Jon Stearley, Ron Oldfield, James H. Laros III, Kevin Pedretti and Ron Brightwell. *Redundant Computing for Exascale Systems*. Sandia National Laboratories, December 2010. Sandia Report SAND2010-8709.

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

[58] 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.

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

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

[61] 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.

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

[63] Ben Vandiver, Hari Balakrishnan, Barbara Liskov, and Sam Madden. Tolerating byzantine faults in transaction processing systems using commit barrier scheduling. *SIGOPS Oper. Syst. Rev.*, 41(6):59–72, October 2007.

[64] 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.

[65] 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.

APPENDIX 4: Facilities and other Resources

Melhem and Znati run two research laboratories located in Sennott Square, which houses The Computer Science Department at the University of Pittsburgh. The labs are fully equipped and furnished and can house 6 researchers, each. Dr. Meneses is a researcher Meneses is a researcher at The Center for Simulation and Modeling at the University of Pittsburgh, where he has fully furnished and equipped office and lab.

Temple CIS (Computer and Information Sciences) department recently moved to a new state of the art building with expanded office, rese arch lab, and instructional lab space. The space contains five instructional labs, each with 40 desktop systems, research labs for mobile computing, robotics, image processing, and high performance computing, and a computing cluster obtained through NSF MRI grant.

APPENDIX 5: Equipment

Melhem and Znati, as well as their students have access to a large number of servers in their laboratories that include clusters of Windows, Mac, and Linux-based workstations. In addition, access is available to the servers owned by the Computer Science Department and the University computing Center. Ethernet and wireless connections are available to every machine in the department. Software licenses for many common software applications are provided by the department without charge for desktop and server machines. The department has four full-time support staffs who act as system administrators and technical engineers. These staffs maintain department and faculty computing resources and facilities. The University's Computing Services and Systems Development (CSSD) also provide a quick access point for professional information technology and computing services 24 hours. This computing environment not only serves the academic computing needs of the campus, but is also becoming the primary means of communication between the students and the faculty. A large number of software packages are available on the school-wide system. Further, as a faculty at the University of Pittsburgh, the PI has access to high-performance computing cycles from the Pittsburgh Supercomputing Center.

The Center for Simulation and Modeling manages a medium-scale computational cluster called Frank. Frank is the central High Performance Computing cluster at the University of Pittsburgh. It is available to everyone on campus. It is designed to cater for a wide range of computational research and is actively used by many departments at Pitt. Frank is divided into two major partitions. The Shared Memory partition supports sequential or multi-threaded jobs which can utilize one or more cores of a single node. The Distributed Memory partition supports jobs that utilize distributed parallelism via MPI or other means, and can make use of 2 or more nodes simultaneously. We present a detailed description of this partition below.

- 40 dual-socket 6-core Intel Westmere (X5650) 2.67 GHz CPU (12 core) nodes and 48 GB of memory. All nodes are connected by QDR InfiniBand.
- 36 dual-socket 8-core Intel Sandy Bridge (E5-2670) 2.6 GHz (16 core) nodes. They have 31 GB of memory, 1 TB of local disk and are connected by QDR InfiniBand.
- 36 dual-socket 8-core Intel Sandy Bridge (E5-2670) 2.6 GHz (16 core) nodes. They have 63 GB of memory, 1 TB of local disk and are connected by QDR InfiniBand.
- 36 quad-socket 16-core AMD Interlagos (Opteron 6276) 2.3 GHz nodes. They have 128 GB of RAM. All nodes are connected by QDR Infiniband and have 2 TB of local scratch.
- 24 quad-socket 16-core AMD Interlagos (Opteron 6276) 2.3 GHz nodes. They have 256 GB of RAM. Both nodes are connected by QDR Infiniband and have 2 TB of local scratch.
- 24 dual-socket 8-core Intel Sandy Bridge (E5-2670) 2.6 GHz (16 core) nodes. They have 128 GB of memory, 1 TB of local disk and are connected by FDR InfiniBand.
- 20 dual-socket 8-core Intel Ivy Bridge (E5-2650v2) 2.6 GHz (16 core) nodes. They have 64 GB of memory, 1 TB of local disk and are connected by FDR InfiniBand.

Department Equipment Summary at Temple University

The computer laboratories at Temple University includes the following:

Hybrid GPU/CPU Cloud Computing Platform

- 72 12-core Intel Xeon nodes with 12GB RAM.
- 6 8-core Intel Xeon nodes with 96GB RAM.
- 6 48-core AMD Opteron nodes w 64GB RAM .
- 12 GPU nodes with 4x Nvidia Tesla C2050
- 108-port switch with 6 leaves
- 108 InfiniBand cards and cables.
- 4 storage nodes 30TB each

(a) TCloud Hybrid GPU/CPU

(b) WiMAX basestation

Figure 8: Temple Computing and Communication Infrastructure

WiMAX Wireless Base Station

- Air4G-W24 2510 Mid (MacroMAXe), 2.56-2.63 GHz, Air4G-W24-2510MT

SGI Altix 3000 Series Computer

- 32 Processors, 32 GB, 2 TB of Memory, NUMA Interconnect

IBM HS20 Cluster (10 Nodes)

- 2 2.4 Ghz CPUs/Node, 2 GB Memory/Node, Gigabit Ethernet Interconnect

Sun X2200 (3 Nodes)

- 8 GB Memory, 2 250 GB SATA Hard Drive, 2 2.0 Ghz AMD Opteron CPUs

Dell PowerEdge 2950 (1 Node)

- 4 GB Memory, 6 143 GB Hard Drive, 2 x 3.0 Ghz Dual-Core CPUs

Dell PowerEdge 1950 (5 Nodes)

- 4 GB Memory, 3 143 GB Hard Drive, 2 3.0 Ghz Dual-Core CPUs

Dell PowerEdge 2850 (4 Nodes)

- 4 GB Memory, 2 73 GB Hard Drive, 2 2.4 Ghz Dual-Core CPUs

Citrix Cluster

- 2 Dell PE 1950 - 2 3 Ghz CPU, 8 GB, 3 143 GB Hard Drive
- 2 Dell SC 1435 - 2 3 Ghz CPU, 4 GB, 2 73 GB Hard Drive
- 1 Dell PE 2950 - 2 3 Ghz CPU, 8 GB, 6 143 GB Hard Drive

Storage and Backup

- Netapp F240c Cluster Storage Server 4 TB
- Promise VTRACK 15100 SCSI Storage; 4 TB, Sun SE3300 SCSI Storage; 2 TB.
- Dell PowerVault TL2000 Tape Backup