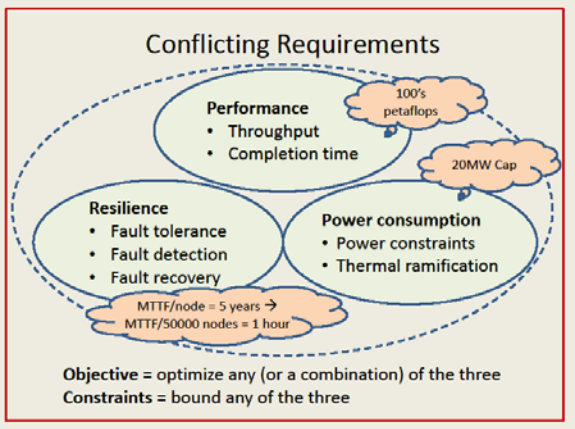
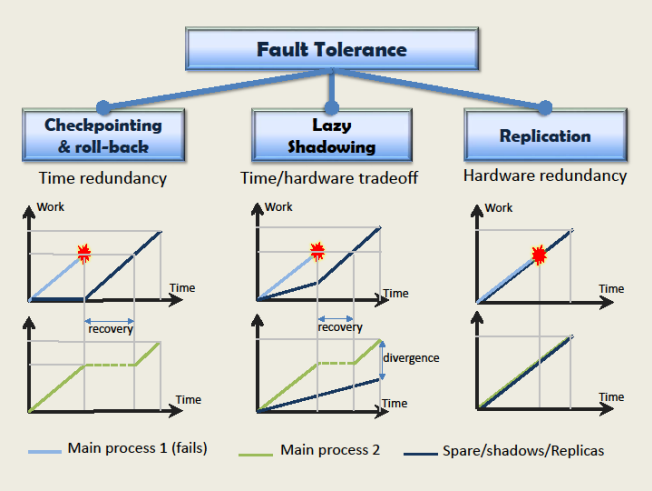
Lazy Shadowing - An Adaptive, Power-Aware, Resiliency Framework for Extreme Scale Computing

Rami Melhem, University of Pittsburgh (Principal Investigator)
Taieb Znati, University of Pittsburgh (Co-Investigator)
Krishna Kant, Temple University (Co-Investigator)

The path to extreme scale computing involves several major road blocks and numerous challenges inherent to the complexity and scale of these systems. A key challenge stems from the stringent requirement, set by the US Department of Energy, to operate in a power envelope of 20 Megawatts. Another challenge stems from the huge number of components, order of magnitudes higher than in existing HPC systems, which will lead to frequent system failures, significantly limiting execution progress. This puts into question the viability of traditional fault-tolerance methods and calls for a reconsideration of the fault-tolerance and power-awareness problem, at scale.

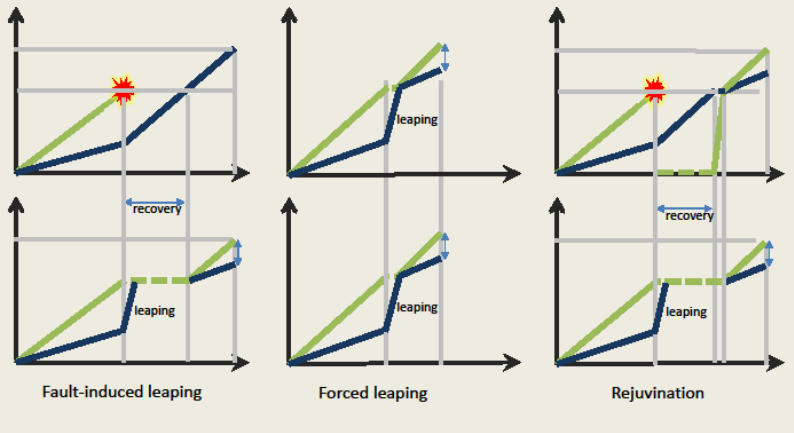




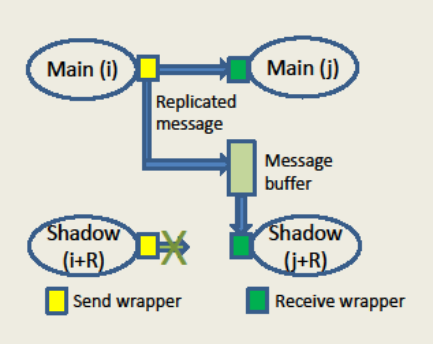
A common approach to resilience relies on time redundancy through checkpointing and rollback recovery. Specifically the execution state is periodically saved to a stable storage, allowing recovery from a failure by restarting from a checkpoint (after rebooting the failed processor or on a spare). As the rate of failure increases, however, the time to periodically checkpoint and rollback leads to a significant drop in efficiency and increase in power. A second approach to resilience is replication, which exploits hardware redundancy by executing simultaneously multiple instances of the same task on separate processors. The physical isolation of processors ensures that faults occur independently, thereby enhancing tolerance to failure. This approach, however, suffers from low efficiency as it dedicates at least 50% of the resources to replicas. This project explores a novel paradigm, ***Lazy Shadowing,*** to achieve high resiliency using a hybrid of time and hardware redundancy. Its basic tenet is to associate with each process a shadow that executes in parallel, but on a different processor and at a reduced rate. The successful completion of the main process causes the immediate termination of the shadow, resulting in significant energy savings over straight replication. When a main process fails, the shadow's execution rate is increased to achieve fast recovery. The reduced initial shadow execution rate can be derived based on the time-to-solution, power constraints, required resiliency and likelihood of failure.

Different approaches can be used to control execution rate, including co-locating multiple shadows on a single processor and/or using Dynamic Voltage and Frequency Scaling. Harnessing the full potential of Lazy Shadowing brings about challenging design problems involving interplay between resilience, forward progress and power. **Shadow Leaping** is used to address these challenges. First, when the shadow of a failed main is recovering, the shadows of the non-failed mains roll forward their execution state to

match that of their main processes, which reduces the recovery time for any subsequent fault because it reduces the divergence between the shadows and their mains. We call this **Fault-induced Leaping**. In the absence of faults, the diversion between mains and shadows can be periodically reduced through **Forced Leaping**. A challenging aspect of Shadow Leaping is in achieving state and communications consistency between main processes and their associated shadows. The project investigates adaptive and energy-efficient approaches to achieve state consistency between main and shadow processes. It also develops efficient algorithms and data structures to maintain state consistency, efficient message logging and correct recovery upon failure. Different design tradeoffs are explored taking into account the fact that increasing the roll-forward frequency and/or increasing the execution rate of the shadows will reduce the size of the message log at the price of reducing the efficiency and/or increasing the power/energy consumption.



When the shadows of *K* main processors are co-located on one processor, the *K+1* processors form a shadowed set. When a fault occur in a main, the execution rate of its shadow is increased by stalling the execution of the other shadows co-located with it. Consequently, the shadowed set becomes vulnerable and may not tolerate any subsequent fault in the set. This vulnerability can be avoided by a **Rejuvenation** process, in which we restart a new process to replace the failed one (after rebooting the failed processor or on a spare) and use leaping to roll forward the new process' state to the correct state recovered by the shadow. At this point the execution rate of the shadow can be reduced again and the execution of the other shadows collocated with it can resume, thus removing the vulnerability of the shadowed set.



In this project, we are constructing a prototype of Lazy Shadowing implemented in MPI. When a user starts *R* ranks, the system transparently creates *R* additional ranks for the shadows and collocates every *K* shadows on a physical processor. An MPI wrapper intercepts MPI calls and conform them to Lazy shadowing. For example, when a rank, *i*, sends a message to rank *j*, the wrapper translates this call to two "send" calls to rank *j* and its shadow at rank *j+R*. The "send" is normally suppressed on shadows. The wrappers also exchange control messages to guarantee timeliness and correctness of recovery. The final implementation will support shadow leaping and rejuvenation.

A program typically computes one or more "state variables" that go into defining the program output. A "program slice" relative to a state variable X, denoted S(X), is the smallest subset of program code that computes X. The size of S(X) depends on to what extent X depends on other state variables. If S(X) is small, executing it in parallel with the original program (along with "acceptance tests" of the results) can significantly accelerate error detection. The proposed research considers how program slicing can be exploited for error detection and enhancing resilience. In particular, lazy Shadowing can be combined with slicing in that the slices that are fraction of their mains can run lazily and provide fault detection capability with reasonable coverage. For example, running at 20% the size of the main program can detect up to 40% of the errors. This project provides compiler techniques to generate slices and the underlying analysis to estimate the corresponding detection coverage. It also explores online learning techniques to estimate the reliability of various parts of an application and accordingly optimize the distribution of slices.