IESP Exascale Challenge: Resilience and Fault Tolerance

Al Geist (ORNL) and Franck Cappello (INRIA)

Research in the reliability and robustness of exascale systems for running large simulations is critical to the effective use of these systems. New paradigms must be developed for handling faults within both the system software and user applications. Hardware support may also be investigated to reduce the fault tolerance overhead. Equally important are new approaches for integrating detection algorithms in both the hardware and software and new techniques to help simulations adapt or be indifferent to faults. One essential element toward these objectives is a better understanding of HPC usage scenarios, applications needs, fundamental origin of the algorithm sensibility to faults, and failures root causes.

What users want is resilience in the execution of their applications. They want to be able to submit a long-running job and have it run to completion in a timely manner. However, because of their scale and complexity, today's supercomputers typically have faults somewhere in the system every day and run for only a few days before the number of faults require rebooting. While supercomputers can often reconfigure around faults, every fault kills the application running on the affected resources. Historically these applications have to be restarted from the beginning or from their last checkpoint, but the checkpoint/restart technique is already losing its effectiveness on petascale systems and will not be viable on exascale systems because of the rate of failures and time required to write out checkpoints. A new fault will occur before the application could be restarted, causing the application to get stuck in a state of constantly being restarted.

Exascale systems will have millions of processors in them and some projections say they will have a billion threads of execution. The major challenge in resilience is that faults in extreme scale systems will be continuous rather than an exceptional event. This requires a major shift from today's software infrastructure. Every layer of the exascale software ecosystem has to be able to cope with frequent faults; otherwise applications will not be able to run to completion. The system software must be designed to detect and adapt to frequent failure of hardware and software components. With the potential that exascale systems will be having constant failures somewhere across the system, application software isn't going to be able to rely on current checkpointing techniques to cope with faults. For exascale systems, new paradigms to tolerate hardware and software faults will need to be developed and integrated into both existing and new applications.

Silent errors are the ìmonster in the closetî for exascale systems. Silent errors are simply faults that occur that never get detected. They can be transient as in the case when a bit or logic gate gets flipped spontaneously. Transient flipping of bits happens continuously in the memory of the largest systems in the world, but ECC memory automatically detects and corrects these faults. Silent errors arise when any part of the memory is not ECC or data paths are not protected, or when multiple memory faults cancel each other out preventing ECC from detecting the faults. Silent errors are not limited to transient affects, for example, an undetected hardware failure is a silent error. Often they are only discovered when the application running on this hardware: gives the wrong answer, fails to complete, or completes much more slowly than usual. By then it is too late for the application to recover. Silent errors are not limited to hardware faults. There have been several cases where software or firmware code has had bugs in it that only manifest in rare cases, for example, router-chip software that changes the bits in one message out of every billion. The key characteristic of silent errors is that they are undetected; therefore, there is no opportunity for an application to adapt or recover from the fault. If the rate of silent errors is too high, then a user must worry that the results of his simulation are correct. This gets back to resilience and correctness of their algorithms and application in the face of faults. Designing mechanisms to tolerate silent errors depend on a better comprehension of these errors especially when they hit the hardware. Very few results are available about the quantitative evaluation of their likelihood at large scale during the application executions.

Exascale systems will need to have much more hardware fault detection built into the architecture and software fault detection built into the software stack in order to reduce the rate of silent errors. Once detected, there is still much work to do, including coordination between different layers of the software stack, deciding on a plan for recovery, reconfiguration, adaptation, and recovery of the application.

When faults become continuous, there will be a critical need for fault oblivious algorithms, and applications that can run-through faults. Very little is known today about how to create such applications except for in the simplest cases that are nearly embarrassingly parallel. The challenge does not rest just with the application developer, the system software also needs to be completely rethought to allow it to cope with a continuous stream of faults and being in a constant state reconfiguration of the system. Much research and paradigm shifts must occur.

In addition to progress in application, system and hardware, there is a need for experimental environments being able to stress and compare different fault tolerance approaches and techniques in a scientific way. Large scale testbeds are essential in the observation and understanding of complex phenomena. Software environments capable of reproducing usage and fault scenarios are also needed to test and debug new resilience concepts at large scale, before putting them in production.