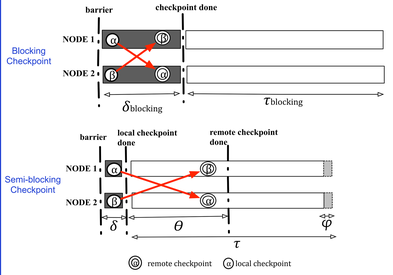
Mitigation of Failures in High Performance Computing via Runtime Techniques

Xiang Ni and Laxmikant V. Kale (advisor), Department of Computer Science, University of Illinois at Urbana-Champaign

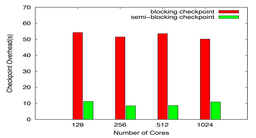
The number of components assembled to create a supercomputer keeps increasing in pursuit of more computational power required to enable breakthroughs in science and engineering. However, the reliability and the capacity of each individual component has not increased as fast as the increase in the total number of components. As a result, the machines fail frequently and hamper smooth execution of high performance applications. This thesis strives to answer the following questions with regard to this challenge: how can a runtime system provide fault tolerance support more efficiently with minimal application intervention? What are the effective ways to detect and correct silent data corruption? Given the limited memory resource, how do we enable the execution and checkpointing of data intensive applications?

Reducing Checkpoint Overhead

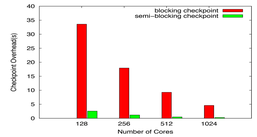
Semi-blocking checkpoint algorithm



Experiments on Trestles at SDSC



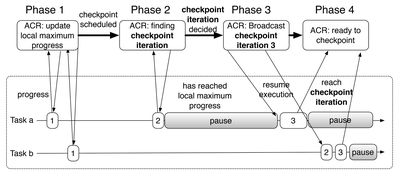
Wave2D weak scaling

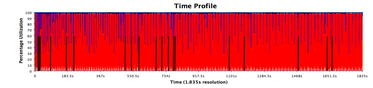


ChaNGa strong scaling

- Resume computation as soon as each node stores its own checkpoint (local checkpoint).
- Interleave the transmission of the checkpoint to buddy with application execution (remote checkpoint).
- To reduce interference, only send out checkpoint messages when there is no application messages ready to be sent.

Automatic checkpoint decision





- Traditionally, application checkpoints at fixed interval.
- What if MTBF follows Weibull process, with increasing or decreasing failure rate?
- How can applications automatically take preventative checkpoint based on failure prediction?

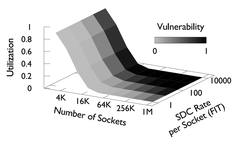
- 19 failures are injected according to Weibull process (shape parameter 0.6) in a 30 minutes run of Jacobi3d.
- Checkpoint period is increased from 6s to 17s automatically by the runtime system.
- Automatic restart: with the support of spare nodes.

Ongoing work

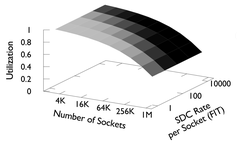
- Simulating the network congestion caused by the coordinated checkpointing with a packet level network simulator: TraceR.
- Can Semi-blocking checkpoint algorithm help achieve faster checkpoint over different network topologies?
- What is the best buddy selection scheme for resilience and performance?
- The effect of optimistic scheduling on applications of different communication-computation ratios.

Detection and Correction of Silent Data Corruption

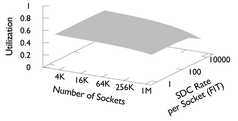
Motivation



No Fault tolerance support
- Low utilization rate
- Cannot protect applications from silent data corruptions.

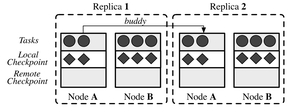


Checkpoint-based protection
- Improved utilization rate
- Cannot protect applications from silent data corruptions.



ACR
- Holistic protection against both silent data corruptions and hard errors.

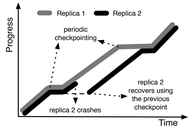
Overview and Optimization



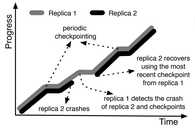
Each node in one replica has a buddy node in the other replica. At checkpoint time, each node first stores a local checkpoint copy and then sends the checkpoint to its buddy node to compare if there is mismatch.

We map the two replicas carefully to reduce the checkpoint overhead.

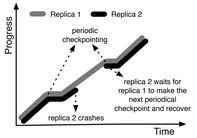
Flexible resilience schemes



Strong resilience
- Replica 2 restarts from the previous checkpoint;
- 100% protection from silent data corruptions.

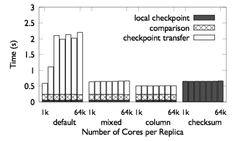


Medium resilience
- Replica 1 performs an immediate checkpoint upon the failure in replica 2;
- Replica 2 recovers from the latest checkpoint;
- Partially unprotected from silent data corruptions.

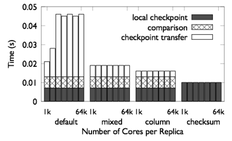


Weak resilience
- Replica 2 restarts from the next checkpoint;
- Unprotected from silent data corruptions for the entire checkpoint interval.

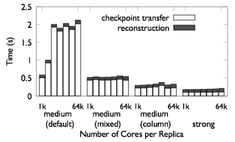
Experiments on Intrepid (IBM BG/P)



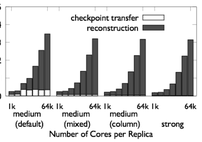
Jacobi3d Checkpoint



LeanMD Checkpoint

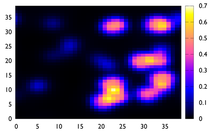


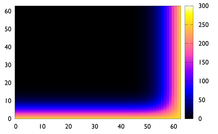
Jacobi3d Restart



LeanMD Restart

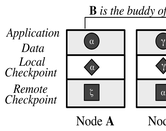
Ongoing work





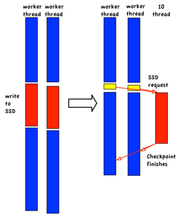
Replication-based scheme provides a baseline scheme to detects all the silent data corruption at the cost of doubling the resource usage. Currently we are exploring cheaper methods to detect SDC automatically based on the attributes of scientific data: spatial smoothness and temporal smoothness.

Relieving Memory Constraints



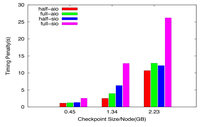
Hard to fit application data and checkpoints in memory at the same time.
Full SSD strategy: store both local and remote checkpoints in SSD
Half SSD strategy: store only remote checkpoint in SSD

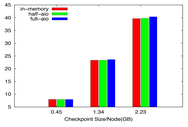
Asynchronous checkpointing to SSD



IO threads
- Write checkpoints to/Read checkpoints from SSD when receiving requests from worker threads;
- Notify worker threads when SSD is done with certain requests.

Experiments on Trestles at SDSC





- Half SSD strategy with asynchronous IO reduces the checkpoint overhead significantly;
- Restarting from SSD does not incur extra overhead.

Ongoing work

For data intensive applications that are hard to fit in memory, we are exploring the use of non-volatile memory as a secondary memory partition with runtime system guidance. The runtime system will transport the data transparently from non-volatile memory to DRAM before it is being used. By learning application access patterns, the runtime system will only offload the infrequently used data to the secondary memory partition to reduce the amount of communication between the two levels memory system.