Application-Transparent Checkpoint/Restart for MPI Programs over InfiniBand

Qi Gao, Weikuan Yu, Wei Huang, Dhabaleswar K. Panda

Network-Based Computing Laboratory
Department of Computer Science & Engineering
The Ohio State University
Introduction

• Nowadays, clusters have been increasing in their sizes to achieve high performance.

• High Performance $\implies$ High Productivity

• Failure rate of the systems grows rapidly along with the system size

• System failures are becoming an important limiting factor of the productivity of large-scale clusters
Motivation

• Most end applications are parallelized
  – Many are written in MPI.
  – More susceptible to failures.
  – Many research efforts, e.g. MPICH-V, LAM/MPI, FT-MPI, C³, etc., for fault tolerance in MPI

• Newly deployed clusters are often equipped with high speed interconnect for high performance
  – InfiniBand: an open industrial standard for high speed interconnect.
    • Used by many large clusters in Top 500 list.
    • Clusters with tens of thousand cores are being deployed

• How to achieve fault tolerance for MPI on InfiniBand clusters to provide both high performance and robustness is an important issue
Outline

• Introduction & Motivation
• Background
  – InfiniBand
  – Checkpointing & rollback recovery
• Checkpoint/Restart for MPI over InfiniBand
• Evaluation framework
• Experimental results
• Conclusions and Future work
InfiniBand

- Native InfiniBand transport services
- Protocol off-loading to Channel Adapter (NIC)
- High performance RDMA operations

InfiniBand Stack (Courtesy from IB Spec.)

- Queue-based model
  - Queue Pairs (QP)
  - Completion Queues (CQ)
- OS-bypass
- Protection & Authorization
  - Protection Domain (PD)
  - Memory Regions (MR) and access keys

Queuing Model (Courtesy from IB Spec.)
Checkpointing & Rollback Recovery

- Checkpointing & rollback recovery is a commonly used method to achieve fault tolerance.
- *Which* checkpointing method is suitable for clusters with high speed interconnects like InfiniBand?
- Categories of checkpointing:

<table>
<thead>
<tr>
<th>Coordinated</th>
<th>Uncoordinated</th>
<th>Communication Induced</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros:</strong></td>
<td><strong>Pros:</strong></td>
<td><strong>Pros:</strong></td>
</tr>
<tr>
<td>Easy to guarantee consistency</td>
<td>No global coordination</td>
<td>Guarantee consistency without global coordination</td>
</tr>
<tr>
<td><strong>Cons:</strong></td>
<td><strong>Cons:</strong></td>
<td><strong>Cons:</strong></td>
</tr>
<tr>
<td>Coordination overhead</td>
<td><em>domino effect</em> or message logging overhead</td>
<td>Requires per-message processing</td>
</tr>
<tr>
<td>All processes must rollback upon failure</td>
<td></td>
<td>High overhead</td>
</tr>
</tbody>
</table>
Checkpointing & Rollback Recovery (Cont.)

• Implementation of checkpointing:

<table>
<thead>
<tr>
<th>System Level</th>
<th>Application Level</th>
<th>Compiler Assisted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros:</strong></td>
<td><strong>Pros:</strong></td>
<td><strong>Pros:</strong></td>
</tr>
<tr>
<td>• Can be transparent to user applications</td>
<td>• Content of checkpoints can be customized</td>
<td>• Application level checkpointing without source code modification</td>
</tr>
<tr>
<td>• Checkpoints initiated independent to the progress of application</td>
<td>• Portable checkpoint file</td>
<td></td>
</tr>
<tr>
<td><strong>Cons:</strong></td>
<td><strong>Cons:</strong></td>
<td><strong>Cons:</strong></td>
</tr>
<tr>
<td>• Need to handle consistency issue</td>
<td>• Applications’ source code need to be rewritten according to checkpointing interface</td>
<td>• Requires special compiler techniques for consistency</td>
</tr>
</tbody>
</table>

• Our current approach: **Coordinated, System-Level, Application Transparent Checkpointing**
Outline

• Introduction & Motivation
• Background
• Checkpoint/Restart for MPI over InfiniBand
• Evaluation Framework
• Experimental Results
• Conclusions and Future Work
Overview

• Checkpoint/Restart for MPI programs over InfiniBand:
  – Using Berkeley Lab’s Checkpoint/Restart (BLCR) for taking snapshots of individual processes on a single node.
  – Design coordination protocol to checkpoint and restart the entire MPI job;
  – Totally transparent to user applications;
  – Does not interfere critical path of data communication.

• Suspend/Reactivate the InfiniBand communication channel in MPI library upon checkpoint request.
  – Network connections on InfiniBand are disconnected
  – Channel consistency is maintained.
  – Transparent to upper layers of MPI library
In our current implementation:

- Process Manager: Multi-Purpose Daemon (MPD), developed in ANL, extended with C/R messaging support
- C/R Library: Berkeley Lab’s Checkpoint/Restart (BLCR)
Global View:
Procedure of Checkpointing

- Initial Synchronization
- Pre-checkpoint Coordination
- Post-checkpoint Coordination
- Local Checkpointing

Data Connections
Data Network

MPI Job Console
Global C/R Coordinator
MPI Process
MPI Process
MPI Process

Control Message Manager
C/R Library
Local C/R Controller
Communication Channel Manager

Checkpoint Request

Running
Global View: Procedure of Restarting

- Process Manager
- Global C/R Coordinator
- MPI Job Console
- MPI Process
- MPI Process
- MPI Process
- Control Message Manager
- C/R Library
- Local C/R Controller
- Communication Channel Manager

Restart Request

Data Connections

Data Network

Restarting → Post-checkpoint Coordination → Running
Local View: InfiniBand Channel in MPI

InfiniBand Host Adapter (HCA)

- QPs
- MRs
- CQs
- PDs

MPI InfiniBand Channel

- Registered User Buffers
- Dedicated Communication Buffers
- Channel Progress Information
- Network Connection Information

InfiniBand Fabric

MPI Upper Layers

- User Application
- MPI InfiniBand Channel

Storage

- Peer MPI Process

Other components include:

- Peer MPI Process
- Peer MPI Process
- Peer MPI Process
- HCA
- HCA
- HCA

Connections and interactions between these components are illustrated in the diagram.
Outline

• Introduction & Motivation
• Background
• Checkpoint/Restart for MPI over InfiniBand
• Evaluation Framework
• Experimental Results
• Conclusions and Future Work
OSU MPI over InfiniBand

- Open Source High Performance Implementations
  - MPI-1 (MVAPICH)
  - MPI-2 (MVAPICH2)
- Has enabled a large number of production IB clusters all over the world to take advantage of InfiniBand
  - Largest being Sandia Thunderbird Cluster (4512 nodes with 9024 processors)
- Have been directly downloaded and used by more than 390 organizations worldwide (in 30 countries)
  - Time tested and stable code base with novel features
- Available in software stack distributions of many vendors
- Available in the OpenFabrics(OpenIB) Gen2 stack and OFED
- More details at
  http://nowlab.cse.ohio-state.edu/projects/mpi-iba/
Evaluation Framework

• Implementation based on MVAPICH2 version 0.9.0
• Will be released with newer version of MVAPICH2 soon
• Test-bed:
  – InfiniBand Cluster with 12 nodes, dual Intel Xeon 3.4 GHz CPUs, 2 GB memory, Redhat Linux AS 4 with kernel version 2.6.11;
  – Ext3 file system on top of local SATA disks
  – Mellanox InfiniHost MT23108 HCA adapters
• Experiments:
  – Analysis of overhead for taking one checkpoint and restart
    • NAS Parallel Benchmarks
  – Performance impact to applications when checkpointing periodically
    • NAS Parallel Benchmarks
    • HPL Benchmark
    • GROMACS
Outline

• Introduction & Motivation
• Background
• Checkpoint/Restart for MPI over InfiniBand
• Evaluation Framework
• Experimental Results
• Conclusions and Future Work
Checkpoint/Restart Overhead

- **Storage overhead**
  - Checkpoint size is same as the memory used by process:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LU.C.8</th>
<th>BT.C.9</th>
<th>SP.C.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkpoint size per process</td>
<td>126MB</td>
<td>213MB</td>
<td>193MB</td>
</tr>
</tbody>
</table>

- **Time for checkpointing**
  - File accessing time is the dominating factor of checkpoint/restart overhead
  - Delay from issuance of checkpoint/restart request to program resumes execution
  - Sync checkpoint file to local disk before program continues

File accessing time is the dominating factor of checkpoint/restart overhead
Performance Impact to Applications
– NAS Benchmarks

- NAS benchmarks, LU, BT, SP, Class C, for 8~9 processes
- For each checkpoint, the execution time increases for about 2-3%
Performance Impact to Applications – HPL Benchmark

- HPL benchmarks, 8 processes.
- Performs same as original MVAPICH2 when taking no checkpoints.
- For each checkpoint, the performance degradation is about 4%.
Benchmarks V.S. Target Applications

• Benchmarks
  – Seconds, minutes (checkpoint in a few minutes)
  – Load all data into memory at beginning
  – The ratio of (memory usage / running time) is high

• Target applications: long running applications
  – Days, weeks, months (checkpoint hourly, daily, or weekly)
  – Computation intensive or load data into memory gradually
  – The ratio of (memory usage / running time) is low

• Benchmarks reflects almost the worst case scenario
  – Checkpointing overhead largely depends on checkpoint file size (process memory usage)
  – Relative overhead is very sensitive to the ratio.
Performance Impact to Applications – GROMACS

- **GROMACS**
  - Molecular dynamics for biochemical analysis.
  - DPPC dataset running on 10 processes.
- Small memory usage with relatively longer running time.
- For each checkpoint, the execution time increases less than 0.3%.
Outline

• Introduction & Motivation
• Background
• Checkpoint/Restart for MPI over InfiniBand
• Evaluation Framework
• Experimental Results
• Conclusions and Future Work
Conclusions and Future Work

• Design & implement a framework to checkpoint and restart for MPI programs over InfiniBand
• Totally transparent to MPI applications
• Evaluations based on NAS, HPL, and GROMACS show that the overhead for checkpointing is not significant

• Future work:
  – Reduce the checkpointing overhead
  – Design a more sophisticated framework for fault tolerance in MPI
  – Integrate into MVAPICH2 release
Acknowledgements

Our research is supported by the following organizations

• Current Funding supported by

  - Office of Science
  - NSF
  - Mellanox Technologies
  - Cisco Systems
  - intel
  - NetworkX
  - NetApp
  - Sun Microsystems

• Current Equipment supported by

  - Mellanox Technologies
  - intel
  - AMD
  - Sun Microsystems
  - APPR
  - Dell
  - Apple
  - advanced clustering technologies, Inc.
  - IBM
  - Microway
  - PathScale
  - SilverStorm
Web Pointers

{gaoq, yuw, huanwei, panda}@cse.ohio-state.edu

http://nowlab.cse.ohio-state.edu/

MVAPICH Web Page
http://nowlab.cse.ohio-state.edu/projects/mpi-iba/