Progress Report on Transparent Checkpointing for Supercomputing

Jiajun Cao, Rohan Garg

College of Computer and Information Science, Northeastern University

{jiajun,rohgarg}@ccs.neu.edu

August 21, 2015
Overview

1. Checkpointing: current status

2. The DMTCP Approach
   - DMTCP and the Software Engineering Burden

3. Towards checkpointing for supercomputing
   - Why are we doing this?
   - Experiments on the MGHPCC Cluster
   - Experiments at Stampede (TACC)
   - Micro-benchmarks: improving performance
   - Experiences while scaling up
Checkpointing: current status

- **Application-specific** – Needs to modify user code; difficult to port to new applications
- **BLCR** – Single-process kernel-level solution; MPI implementations and resource managers need to add extra checkpoint-restart modules; requires tuning for different kernel versions
- **CRIU for Containers** – Linux kernel configured to expose certain of its internals in user-space; difficult to use with MPI
- **DMTCP** – No need to modify the OS or the user program; transparent support for distributed applications (TCP/InfiniBand)
The DMTCP Approach

- Direct support for distributed applications “out-of-the-box”
- Entirely in user-space, no root privileges
- Transparently inter-operates with:
  1. Major MPI implementations: (MVAPICH2, Open MPI, Intel MPI, MPICH2)
  2. Resource managers (e.g., SLURM, Torque, planned for LSF)
     - Resource Manager plugin (batch-queue)
  3. High-performance networks
     - InfiniBand plugin
  4. MPI process managers: (e.g., Hydra, PMI, mpispawn, ibrun)
  5. All recent versions of Linux kernel
- Extensible: supports application-specific plugins
  - A “cut-out” plugin — if memory is zero, don’t save
  - Checkpoint only when specified by the application
DMTCP and the Software Engineering Burden

**Lines of code**: a proxy for the required software effort

- **DMTCP**
  - C/C++: 22,276 (+ 1657 for comments)
  - Headers: 6,693 (+ 2173 for comments)

- **InfiniBand plugin**
  - C/C++: 8,500 (+ 705 for comments)
  - Headers: 342 (+ 229 for comments)

- **RM plugin**
  - C/C++: 3,859 (+ 451 for comments)
  - Headers: 591 (+ 753 for comments)

* All measurements using `cloc`
Why are we doing this?

DMTCP-style transparent checkpointing is used with full memory dumps.

1990s (Beowulf cluster):
one disk per node; full memory dump is efficient

2000s (Back-end storage):
more RAM per compute node; many compute nodes writing to a small number of storage nodes, full memory dumps become less efficient

Future (Local SSD storage):
a few compute nodes writing to one SSD; full memory dump is efficient

Conclusion: DMTCP-style checkpointing on supercomputers will be practical in the future. The time to plan for this is now!
Experiments on MGHPCC: Runtime overhead

MGHPCC: Massachusetts Green High-Performance Computing Center, http://www.mghpcc.org/

Figure: Comparison of runtime overhead (Open MPI); for jobs requiring at least 50 seconds the runtime overhead was small (< 2%); for jobs requiring less than 50 seconds, the DMTCP startup time was a significant fraction of the total.

Jiajun Cao, Rohan Garg (NU)
Experiments on MGHPC: Checkpoint overhead

<table>
<thead>
<tr>
<th>NAS benchmark</th>
<th>Number of processes</th>
<th>Ckpt time (s)</th>
<th>Ckpt size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU.E 128×4</td>
<td>70.8</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>LU.E 64×8</td>
<td>136.6</td>
<td>356</td>
<td></td>
</tr>
<tr>
<td>LU.E 32×16</td>
<td>222.6</td>
<td>355</td>
<td></td>
</tr>
<tr>
<td>LU.E 128×16</td>
<td>70.2</td>
<td>117</td>
<td></td>
</tr>
</tbody>
</table>

Table: Checkpoint times and image sizes for the same NAS benchmark, under different configurations. The checkpoint image size is for a single MPI process. The checkpoint time is roughly proportional to the total size of the checkpoint images on a single node.

NOTE: These were preliminary results with Open MPI ver. 1.6 with a NFS shared filesystem.
Experiments at Stampede (TACC): Runtime overhead

Latency

- w/o DMTCP
- w/o IB
- DMTCP + IB

Bandwidth

- w/o DMTCP
- w/o IB
- DMTCP + IB

Bidirectional Bandwidth

- w/o DMTCP
- w/o IB
- DMTCP + IB

Multiple Bandwidth

- w/o DMTCP
- w/o IB
- DMTCP + IB
### Experiments at Stampede (TACC): Runtime overhead

<table>
<thead>
<tr>
<th>NAS benchmark</th>
<th>Number of processes</th>
<th>Native (s)</th>
<th>with DMTCP (s)</th>
<th>Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU.C</td>
<td>2</td>
<td>354.7</td>
<td>356.7</td>
<td>0.6</td>
</tr>
<tr>
<td>LU.C</td>
<td>4</td>
<td>179.9</td>
<td>180.5</td>
<td>0.3</td>
</tr>
<tr>
<td>LU.C</td>
<td>8</td>
<td>95.3</td>
<td>95.8</td>
<td>0.5</td>
</tr>
<tr>
<td>LU.D</td>
<td>16</td>
<td>1104.0</td>
<td>1104.9</td>
<td>0.0</td>
</tr>
<tr>
<td>LU.D</td>
<td>32</td>
<td>497.0</td>
<td>508.4</td>
<td>2.2</td>
</tr>
<tr>
<td>LU.D</td>
<td>64</td>
<td>293.5</td>
<td>296.5</td>
<td>1.0</td>
</tr>
<tr>
<td>LU.D</td>
<td>128</td>
<td>156.9</td>
<td>159.0</td>
<td>1.3</td>
</tr>
<tr>
<td>LU.E</td>
<td>256</td>
<td>1604.8</td>
<td>1635.2</td>
<td>1.9</td>
</tr>
<tr>
<td>LU.E</td>
<td>512</td>
<td>619.3</td>
<td>628.9</td>
<td>1.6</td>
</tr>
<tr>
<td>LU.E</td>
<td>1024</td>
<td>535.3</td>
<td>544.6</td>
<td>1.7</td>
</tr>
<tr>
<td>LU.E</td>
<td>2048</td>
<td>168.7 (162.0)</td>
<td>184.7 (165.4)</td>
<td>9.5 (2.0)</td>
</tr>
<tr>
<td>LU.E</td>
<td>4096</td>
<td>94.4 (86.7)</td>
<td>167.5 (91.3)</td>
<td>77.4 (5.0)</td>
</tr>
</tbody>
</table>

**Table:** Runtime overhead for the NAS LU benchmark. MVAPICH-1.9 with the MV2_ON_DEMAND_THRESHOLD environment variable was used up to 1024 cores. For 2048 cores and for 4096 cores MVAPICH-2.1 was used. The numbers in the parentheses are the actual computation times as reported by the benchmark.
Experiments at Stampede (TACC): Checkpoint overhead

Checkpoint times are approximately proportional to the image size. For a given problem class the image size decreases with the number of processes.

<table>
<thead>
<tr>
<th>NAS benchmark</th>
<th>Number of processes</th>
<th>Ckpt time (s)</th>
<th>Ckpt size (MB)</th>
<th>Restart time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU.C</td>
<td>2</td>
<td>16.0</td>
<td>330</td>
<td>7.5</td>
</tr>
<tr>
<td>LU.C</td>
<td>4</td>
<td>8.6</td>
<td>174</td>
<td>4.3</td>
</tr>
<tr>
<td>LU.C</td>
<td>8</td>
<td>5.0</td>
<td>98</td>
<td>3.2</td>
</tr>
<tr>
<td>LU.D</td>
<td>16</td>
<td>32.0</td>
<td>650</td>
<td>12.9</td>
</tr>
<tr>
<td>LU.D</td>
<td>32</td>
<td>16.9</td>
<td>350</td>
<td>8.1</td>
</tr>
<tr>
<td>LU.D</td>
<td>64</td>
<td>9.9</td>
<td>190</td>
<td>6.5</td>
</tr>
<tr>
<td>LU.D</td>
<td>128</td>
<td>6.1</td>
<td>115</td>
<td>4.9</td>
</tr>
<tr>
<td>LU.E</td>
<td>256</td>
<td>33.8</td>
<td>700</td>
<td>18.9</td>
</tr>
<tr>
<td>LU.E</td>
<td>512</td>
<td>19.6</td>
<td>370</td>
<td>19.1</td>
</tr>
<tr>
<td>LU.E</td>
<td>1024</td>
<td>12.4</td>
<td>210</td>
<td>11.3</td>
</tr>
<tr>
<td>LU.E</td>
<td>2048</td>
<td>(in progress)</td>
<td>(in progress)</td>
<td>(in progress)</td>
</tr>
<tr>
<td>LU.E</td>
<td>4096</td>
<td>(in progress)</td>
<td>(in progress)</td>
<td>(in progress)</td>
</tr>
</tbody>
</table>

Table: Checkpoint times and image sizes for the NAS LU benchmark.
Micro-benchmarks: improving performance

- Performance bug in DMTCP found through micro-benchmarks (valloc)
- Performance bug in the IB plugin found through micro-benchmarks (malloc): a fix is planned
Experiences while scaling up

4 cores:
- Lustre FS bug

32 cores:
- Multiple processes on a compute node: fixes to DMTCP’s code that handles shared memory segments

64 cores:
- Raised value of MV2_ON_DEMAND_THRESHOLD

128 cores:
- PMI external socket: added it to the list of “external” sockets

2048 cores:
- Initialization too slow with MVAPICH-1.9: switch to MVAPICH-2.1

16,384 cores:
- ???
Questions?