Checkpointing the Un-checkpointable: MANA and the Split-Process Approach

Gene Cooperman*

gene@ccs.neu.edu

Khoury College of Computer Sciences
Northeastern University, Boston, USA

August 20, 2019

* Partially supported by NSF Grants ACI-1440788 and OAC-1740218, and by grants from Intel Corporation and Mentor Graphics (a division of Siemens).
Table of Contents

1. DMTCP — A review
2. DMTCP Plugins — A review
3. Checkpointing with Proxies: Beginnings of a New Paradigm
4. Split Processes: MANA for MPI
Outline

1. DMTCP — A review
2. DMTCP Plugins — A review
3. Checkpointing with Proxies: Beginnings of a New Paradigm
4. Split Processes: MANA for MPI
The DMTCP project and antecedents began almost 15 years ago, and is widely used today:
http://dmtcp.sourceforge.net/publications.html

**Typical use case for HPC:** 12-hour batch time slot, an application is expected to finish in 18 hours.
(NOTE: A resource manager such as SLURM can send a signal one hour before the end of the time slot, giving the application time to checkpoint; DMTCP can then transparently checkpoint the state.)

Ease of use (unprivileged and transparent):
```
dmtcp_launch a.out arg1 arg2 ...

dmtcp_command --checkpoint # from other terminal or window

dmtcp_restart ckpt_a.out_*.dmtcp
```

( DMTCP also works for programs that are multi-threaded, distributed, MPI-based, ....)
DMTCP Architecture: Coordinated Checkpointing
Fundamental Research Question for the DMTCP Team

“What are the limits of checkpointing applications in the real-world?”

**PROBLEM:** An application may store a session id, tty, network peer address, TMPDIR for temporary files, process id, thread id, etc. On restart, some or all of these are likely to change.

**SOLUTION:** Process virtualization: interpose on all calls to such ids; replace actual id by DMTCP-defined virtual id.

**PRINCIPLE:** “Never let the application see a real id!”

**Scalability:**
Checkpointing HPCG: 32,752 CPU cores), and (NAMD: 16,368 CPU cores) for checkpointing MPI applications natively over InfiniBand at TACC: “System-level Scalable Checkpoint-Restart for Petascale Computing”, Jiajun Cao et al., Int. Conf. on Parallel and Dist. Sys. (ICPADS’16), 2016

("To the best of our knowledge, this is 100 times larger than the previously largest transparent checkpointing study in the literature.")
WHY PLUGINS?

- Processes must talk with the rest of the world!
- *Process virtualization:* virtualize the connections to the rest of the world

In short, a plugin is responsible for modeling an external subsystem, and then creating a semantically equivalent construct at the time of restart.
PRINCIPLE:
The user sees only virtual pids; The kernel sees only real pids

<table>
<thead>
<tr>
<th>PID: 4000</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PID: 4001</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>4001</td>
<td></td>
</tr>
</tbody>
</table>

Translation Table

<table>
<thead>
<tr>
<th>Virt. PID</th>
<th>Real PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>2652</td>
</tr>
<tr>
<td>4001</td>
<td>3120</td>
</tr>
</tbody>
</table>

getpid() 2652

kill(4001, 9)

4001

Sending signal 9 to pid 3120

KERNEL

3120

Fictional scenario with ball-park numbers (no particular vendor):
- Software circuit simulation: about 1 million times slowdown
- Hardware emulation at back-end: about 1 thousand times slowdown
- Cost of back-end hardware emulator: about $800,000
- **Use case A** (for a new CPU design): Boot Microsoft Windows overnight with emulator, and then test Microsoft Office.
- **Use case B**: Boot Microsoft Windows overnight with emulator, and checkpoint. In later iterations, restart, and then test Microsoft Office.

The above fictional scenario requires a DMTCP plugin to model the back-end emulator. *See publications with emulator vendors for details.*
GOAL: *To convince you of a general proxy paradigm for handling the “hard” checkpointing challenges that remain.*

We take as testbeds two such “hard” examples:

1. **Checkpoint a CUDA application running on a GPU** (Problem: how to save and restore the state of GPU hardware) (joint with Rohan Garg, Apoorve Mohan, Michael Sullivan)
   
   To appear, IEEE Cluster’18; see, also, technical report: https://arxiv.org/abs/1808.00117

2. **Checkpoint MPI on NERSC/Cori supercomputer** (with GNI network; no checkpoint-restart service to support it)
   
   (a long story: coming next in Part Two of this talk)
Proxies: The Secret Sauce

BEFORE:

GPU

CUDA LIBRARY

Application

APPLIC.

AFTER:

GPU

CUDA LIBRARY

Application

Process

Proxy

Process

INTERPOSE

APPLIC.

CUDA LIB
Basic Idea of a Proxy for System Services

**PROBLEM:** Often, system services are provided with the help of an auxiliary system that cannot be checkpointed.

**EXAMPLES:** GPU device hardware; MPI auxiliary process or MPI coordinator; sshd (ssh daemon); VNC server; etc.

**SOLUTION:**

A. Split user process into two: an application process with all of the application state; and a proxy process communicating with hardware or software for system services.

B. System service requests are passed from application process to proxy process through inter-process communication, and pass result back.

C. At checkpoint time, the proxy process is temporarily disconnected, and the application process is checkpointed as an isolated “vanilla” Linux process. (Note: the proxy process must be in a quiescent state (no unfinished system service tasks) at checkpoint time.)

D. At restart time, a *new* proxy process re-connects with the system service and with the restarted application. The application process then replays some old system service requests, restoring system to a state that equivalent to pre-checkpoint time.

For more on this, see the thesis of Rohan Garg

(NOTE: Proxies have been used before. For example, this is the basis of an old trick for checkpointing VNC sessions. We propose it here as a general paradigm for checkpoint-restart.)
BEFORE:

GPU

CUDA LIBRARY

APPLIC.

Application

AFTER:

GPU

CUDA LIBRARY

APPLIC.

INTERPOSE

CUDA LIB

Application Process

Proxy Process

*First tell them what you’re going to say; then say it; and then tell them what you said.
Proxy-based shadow paging for CUDA UVM (Unified Virtual Memory)

- **Shadow UVM page synchronization**: Catches memory transfers between proxy and application through memory permissions and segfault detection. The difficulty for transparent checkpointing with CUDA-managed memory: How to make this efficient?

  See the CRUM paper (Algorithm 1, shadow-page synchronization) for details.

- Enables fast *forked checkpointing model* for UVM memory that overlaps writing a checkpoint image to stable storage, while the application continues. *Almost free benefit of our approach!* (This was difficult in the past due to the need to share memory among the GPU device, and the UVM-based host that was split among parent and forked child processes.)
CRUM (Checkpoint-Restart for CUDA Unified Memory): Runtime

Figure: Runtime overheads for different benchmarks under CRUM.

Operating environment: Four NVIDIA Tesla P100’s per node; CUDA 8 (max config: four nodes with 4 GPUs per node and 8 MPI ranks per node)
1. DMTCP — A review
2. DMTCP Plugins — A review
3. Checkpointing with Proxies: Beginnings of a New Paradigm
4. Split Processes: MANA for MPI
MANA for MPI: MPI-Agnostic Network-Agnostic Transparent Checkpointing

Full paper with details at:

Not just an implementation, but a flexible principle with many applications:

- **IDEA:** Load two independent programs into a single process, sharing a single address space. (For a different approach, see “Process-in-process”, HPDC’18 from RIKEN et al., employing multiple link maps/dlmopen.)

- **LOW OVERHEAD!** This completely eliminates the overhead of the proxy approach. (No need for shared memory, cross-memory-attach (cma), XPMEN. One program can directly pass an internal pointer to the other program.)

- **Isolate the MPI/network libraries into their own program; completely separate from the program running the MPI application.**
FEATURES OF THE SPLIT-PROCESS APPROACH:

- Can dynamically change configuration of underlying MPI at runtime.  
  *(Why not? The MPI libraries run in a separate program, unrelated to the MPI application program.)*

- Can even change the choice of the underlying MPI and network  
  (e.g., InfiniBand vs. Cray GNI) at runtime!  
  *(Why not? The MPI and network libraries run in a separate program, unrelated to the MPI application program.)*

- Can even migrate to a new cluster at runtime, in which the number of  
  CPU cores per nodes is different!  
  *(Why not? The binding of the MPI libraries to the CPU cores is part of a separate program, unrelated to the MPI application program.)*
Puzzle

Can you solve checkpointing on...

Cray MPI over Infiniband

And restart on...

MPICH over TCP/IP

4 Nodes, 4 Cores/Ranks per Node

8 Nodes, 2 Cores/Ranks per Node

Shared Memory

Shared Memory
Cross-Cluster Migration

It is now possible to checkpoint on Cray MPI over Infiniband
And restart on MPICH over TCP/IP

- 4 Nodes, 4 Cores/Ranks per Node
- 8 Nodes, 2 Cores/Ranks per Node

Shared Memory

Gene Cooperman
Checkpointing: MANA, Split-Process Approach
August 20, 2019
Transparency and Agnosticism

Transparency

1. No re-compilation and no re-linking of application
2. No re-compilation of MPI
3. No special transport stack or drivers

Agnosticism

1. Works with any libc or Linux kernel
2. Works with any MPI implementation (MPICH, CRAY MPI, etc)
3. Works with any network stack (Ethernet, Infiniband, Omni-Path, etc).
Alas, poor transparency, I knew him Horatio...

Transparent checkpointing could die a slow, painful death.

1. Open MPI Checkpoint-Restart service (Network Agnostic; cf. Hursey et al.)
   ○ MPI implementation provides checkpoint service to the application.
2. BLCR
   ○ Utilizes kernel module to checkpoint local MPI ranks
3. DMTCP (MPI Agnostic)
   ○ External program that wraps MPI for checkpointing.

These, and others, have run up against a wall:

MAINTENANCE
The M x N maintenance penalty

MPI:
- MPICH
- OPEN MPI
- LAM-MPI
- CRAY MPI
- HP MPI
- IBM MPI
- SGI MPI
- MPI-BIP
- POWER-MPI
- ....

Interconnect:
- Ethernet
- InfiniBand
- InfiniBand + Mellanox
- Cray GNI
- Intel Omni-path
- libfabric
- System V Shared Memory
- 115200 baud serial
- Carrier Pigeon
- ....
The M x N maintenance penalty

MPI:
- MPICH
- OPEN-MPI
- LAM-MPI
- CRAY MPI
- HP MPI
- IBM MPI
- SGI MPI
- MPI-BIP
- POWER-MPI
- ...

Network Agnostic

Interconnect:
- Ethernet
- InfiniBand
- InfiniBand + Mellanox
- Cray GNI
- Intel Omni-path
- libfabric
- System V Shared Memory
- 115200 baud serial
- Carrier Pigeon
- ...

Gene Cooperman
Checkpointing: MANA, Split-Process Approach
August 20, 2019
The M x N maintenance penalty

MPI:
- MPICH
- OPEN-MPI
- LAM-MPI
- CRAY MPI
- HP MPI
- IBM MPI
- SGI MPI
- MPI-BIP
- POWER-MPI
- ...

MPI and Network Agnostic

Interconnect:
- Ethernet
- InfiniBand
- InfiniBand + Mellanox
- Cray GNI
- Intel Omni-path
- libfabric
- System V Shared Memory
- 115200 baud serial
- Carrier Pigeon
- ...

Gene Cooperman
Checkpointing: MANA, Split-Process Approach
August 20, 2019
MANA: MPI-Agnostic, Network-Agnostic

The problem stems from checkpointing both the MPI coordinator and the MPI lib.
MANA: MPI-Agnostic, Network-Agnostic

The problem stems from checkpointing MPI - both the coordinator and the library.
Achieving Agnosticism

Step 1: Drain the Network

As demonstrated by Hursey et al., abstracting by “MPI Messages” allows for Network Agnosticism.

Chandy-Lamport Algorithm
Checkpointing Collective Operations

Solution: Two-phase collectives

1. Preface all collectives with a trivial barrier
2. When the trivial barrier is completed, call the original collective

Gene Cooperman
Checkpointing Collective Operations

Solution: Two-phase collectives

1. Preface all collectives with a trivial barrier
2. When the trivial barrier is completed, call the original collective
Achieving Agnosticism

Step 2: Discard the network
Checkpointing A Rank

Solution: Isolation

Checkpointing the rank is simpler... right?

Problems:
- MPI Implementation Specific
- Contains MPI network state
- Required by MPI and Application
- Platform dependant

MPI Rank

MPI Application

MPI Library

Network Libraries

LIBC

- Grouping information
- Opaque MPI Objects
- Heap Allocations
Isolation - The “Split-Process” Approach

Terminology

Upper-Half program

Lower-Half program

Single Memory Space

MPI Application

MPI Proxy Library

MPI Library

Network Libraries

LIBC

Checkpoint and Restore

Standard C Calling Conventions
No RPC involved

Discard and Re-initialize
MPI Agnosticism Achieved

Upper Half:
Persistent Data

Lower Half:
Ephemeral Data

MPI Application
Config and Drain Info
LIBC

*Special care must be taken when replacing upper half libraries

Lower half data can be replaced by new and different implementations of MPI and related libraries.
Checkpoint Process

Step 1: Drain the Network
Checkpoint Process

Step 1: Drain the Network
Step 2: Checkpoint Upper-Half

- MPI Rank
- MPI Application
- Config and Drain Info
- LIBC
Restart Process

Step 1: Restore Lower-Half

Lower-half components may be replaced
Restart Process

Step 1: Restore Lower-Half
Step 2: Re-initialize MPI

- MPI_INIT
- Replay Configuration

Naturally       Optimized

MPI Library
MPI Proxy Library
Network Libraries
LIBC

Lower-half components may be replaced
Restart Process

Step 1: Restore Lower-Half
Step 2: Re-initialize MPI
Step 3: Restore Upper-Half

MPI Rank # assigned by MPI_Init
used to select checkpoint file for
restoring the upper half.

This avoids the need to virtualize
MPI Rank numbers.

Lower-half components may be replaced

- MPI_INIT
- Replay Configuration

Gene Cooperman
Checkpointing: MANA, Split-Process Approach
August 20, 2019
Puzzle

Can you solve checkpointing on...
Cray MPI over Infiniband
And restart on...
MPICH over TCP/IP

4 Nodes, 4 Cores/Ranks per Node
8 Nodes, 2 Cores/Ranks per Node

YES
Checkpoint-Restart Overhead

Checkpoint Data Size

- GROMACS - 64 Ranks over 2 Nodes: 5.9GB (and 0.6% runtime overhead)
- HPCG - 2048 ranks over 64 nodes: 4TB (and nearly 0% runtime overhead)
- Largely dominated by memory used by benchmark program.

Checkpoint Time

- Largely dominated by disk-write time
- “Stragglers” - a single rank takes much longer to checkpoint than others.

Restart Time

- MPI state reconstruction represented < 10% of total restart time.
NEW: Cross-Cluster MPI Application Migration

Traditionally, migration across disparate clusters was not feasible.

- Different MPI packages across clusters
- Highly optimized configurations tied to local cluster (Caches, Cores/Node)
- Overhead of checkpointing entire MPI state is prohibitive

Overhead of migrating under MANA:

- 1.6% runtime overhead after migration.*

* Linux kernel’s upcoming patch https://lwn.net/Articles/769355/ reduces overhead to 0.6%
Outline

1. DMTCP — A review
2. DMTCP Plugins — A review
3. Checkpointing with Proxies: Beginnings of a New Paradigm
4. Split Processes: MANA for MPI
1 *If we haven’t truly begun …*

Maybe some processes have not completed their current computation task. Maybe we’ll wait a long time before they reach the collective communication. Maybe we should abort the collective communication, and checkpoint immediately.

(*Aborting the collective communication should be safe. We hope that the lower-half MPI didn’t start writing yet into the upper user buffers that were passed as arguments.*)

2 *But maybe we have begun and we’re in the middle of it …*

Maybe all processes have already reached the collective communication, and some of them have begun write persistent changes into the upper half user buffers that were passed in. It’s dangerous to abort if we’ve written to the user’s buffer. So, maybe we should finish the collective communication if we’ve truly begun it. We can checkpoint after that.

*What to do?? (I’m so confused. 😞)*
Checkpointing Collective Operations

Solution: Two-phase collectives

1. Preface all collectives with a trivial barrier
2. When the trivial barrier is completed, call the original collective
Collective Communication: How can it work?

1. Maybe some processes are still in the trivial barrier. But then no processes are in the actual barrier invoked by the application.
   - **Solution:** At restart time, discard any operations in the trivial barrier from the lower half, and restart the trivial barrier in the new (restarted) lower half MPI library.

2. Maybe some processes are still in the actual barrier invoked by the application. But then no processes are in the trivial barrier.
   - **Solution:** We know that all processes must have reached the collective communication, since they have passed through the trivial barrier. So, just wait until they finish the actual collective communication invoked by the application.
   
   *There will be no delay. All processes have entered the collective communication!*

(But see the paper for the formal details:)


*High Performance Distributed Computing* (HPDC’19)
THANKS TO THE MANY STUDENTS AND OTHERS WHO HAVE CONTRIBUTED TO DMTCP OVER THE YEARS:

Jason Ansel, Kapil Arya, Alex Brick, Jiajun Cao, Tyler Denniston, Xin Dong, William Enright, Rohan Garg, Paul Grosu, Twinkle Jain, Samaneh Kazemi, Jay Kim, Gregory Kerr, Apoorve Mohan, Mark Mossberg, Gregory Price, Manuel Rodríguez Pascual, Artem Y. Polyakov, Michael Rieker, Praveen S. Solanki, Ana-Maria Visan

QUESTIONS?