Extending DMTCP Checkpointing for a Hybrid Software World

Gene Cooperman
gene@ccs.neu.edu

College of Computer and Information Science
Northeastern University, Boston, USA
and Université Fédérale Toulouse Midi-Pyrénées

August 16, 2017

[*] Partially supported by NSF Grant ACI-1440788, by a grant from Intel Corporation, and by an IDEX Chaire d'Attractivité (Université Fédérale Toulouse Midi-Pyrénées) under Grant 2014-345.

Table of Contents

Outline

DMTCP: A Demo

```
DMTCP% vi test/dmtcp1.c
> int main(int argc, char* argv[])
> { int count = 1;
>   while (1)
>   {     printf(" %2d ",count++);
>    fflush(stdout);
>    sleep(2); }
>   return 0; }
DMTCP% test/dmtcp1
  1   2   3 ^C
DMTCP% bin/dmtcp_launch --interval 5 test/dmtcp1
  1   2   3   4   5   6   7 ^C
DMTCP% ls ckpt_dmtcp1*
ckpt_dmtcp1_66e1c8437adb789-40000-5745d372.dmtcp
DMTCP% bin/dmtcp_restart ckpt_dmtcp1*
  7   8   9  10 ^C
```

DMTCP: A First Look

DMTCP: Distributed MultiThreaded CheckPointing

- As easy to use as:

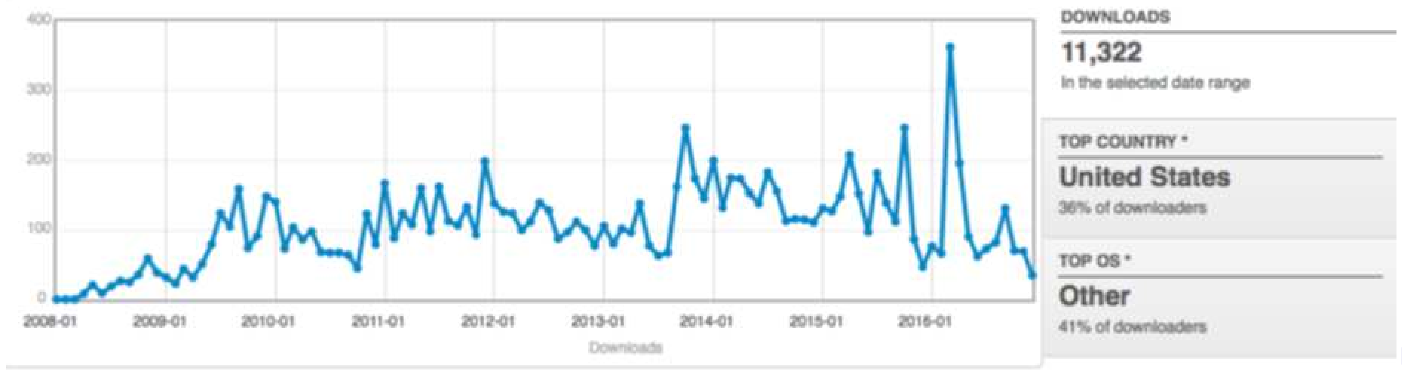
```
dmtcp_launch ./myapp
dmtcp_command --checkpoint
dmtcp_restart ckpt_myapp_*.dmtcp
```

- and DMTCP is contagious: It follows `fork()`, `ssh`, etc.

Free and Open Source: `http://dmtcp.sourceforge.net`

The DMTCP project is now in its second decade.

- Published literature: more than 50 other groups (not us).
  `http://dmtcp.sourceforge.net/publications.html`
- *Downloads:*



DOWNLOADS
11,322
In the selected date range

TOP COUNTRY *
United States
36% of downloaders

TOP OS *
Other
41% of downloaders

What is Checkpointing?

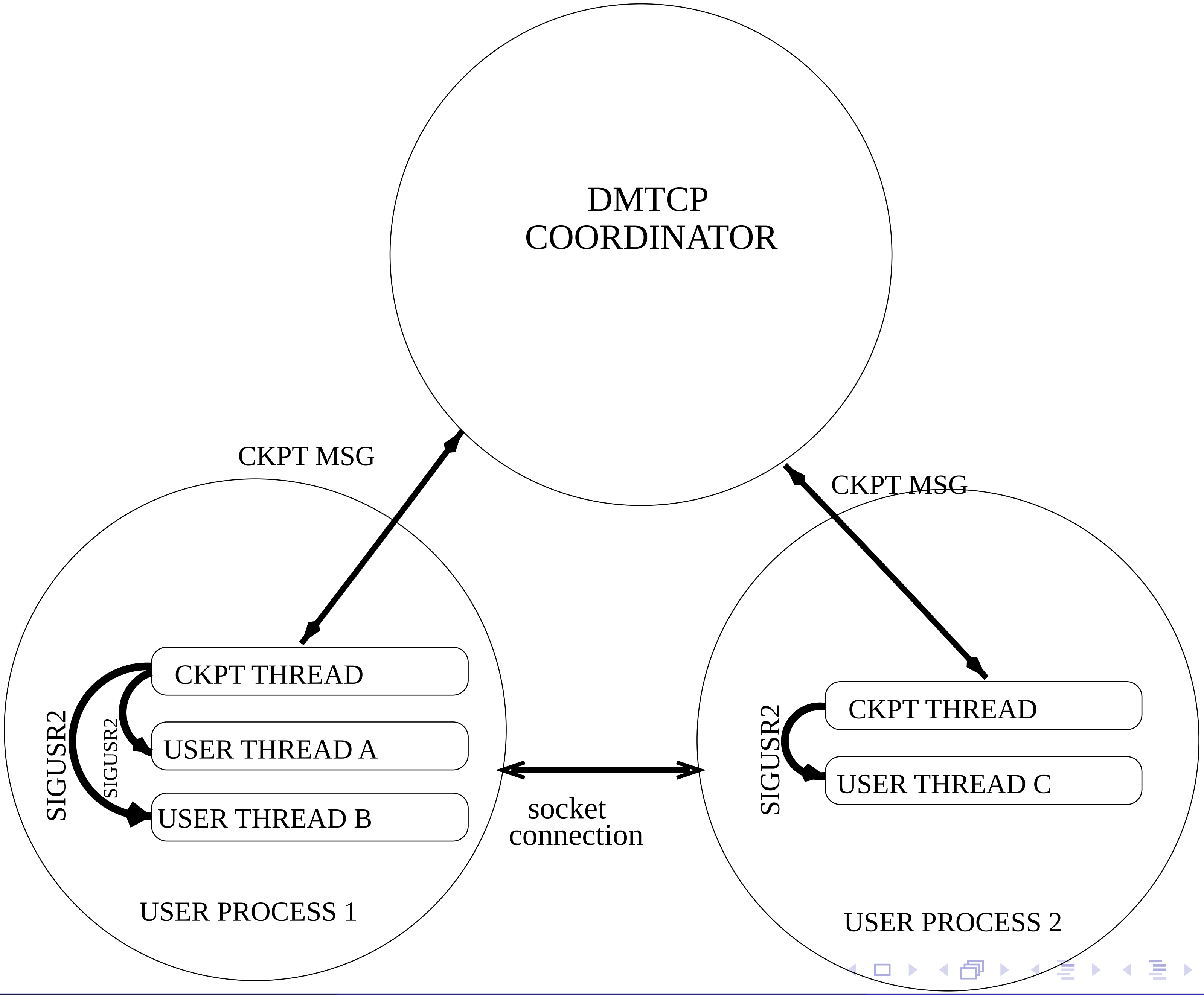
Checkpointing is the action of saving the state of a running process to a checkpoint image file.
Checkpointing supports several other features for free!

1. *Process migration* is the action of migrating a running process from one computer to a different computer.
   Process migration is easy: just copy the checkpoint image file to a new computer, and restart there.
2. *Process replication* is the action of creating a copy of a running process.
   Process replication is easy: just copy the checkpoint image file to a new computer or directory, and restart both the original and the copy of the checkpoint image file.

Uses for Checkpointing

1. Fault tolerance (if the process crashes, then roll back to a previous checkpoint)
2. Extended sessions (if it's time to go home to dinner, then checkpoint and restart the next day)
3. Debugging (checkpoint every 30 seconds; if the process crashes, restart from the last checkpoint under a debugger, and analyze)
4. Reproducible Bug Reports (checkpoint every 30 seconds; if the process crashes, submit the last checkpoint image to the program developer)
5. Fast startup of a process (checkpoint after the process starts, and then restart from the ckpt image file in the future)

DMTCP Architecture: Coordinated Checkpointing



Principles

- **One DMTCP coordinator = one (checkpointable) DMTCP comput.;** Can have multiple coordinators/computations separately checkpointable
- Either the DMTCP checkpoint thread is active or the user thread, but not both at the same time.
- No single point of failure, providing that checkpoint image files are backed up: Even if the coordinator dies, just restart from last checkpoint.
- The runtime libraries are saved as part of the memory image. So, the application continues to use the same library API.
- The Linux environment variables are part of the memory image. (A special DMTCP plugin must be invoked to change any environment variables that were saved at the time of checkpoint.)
- Everything is in user-space; no admin privileges needed.

Outline

DMTCP Plugins

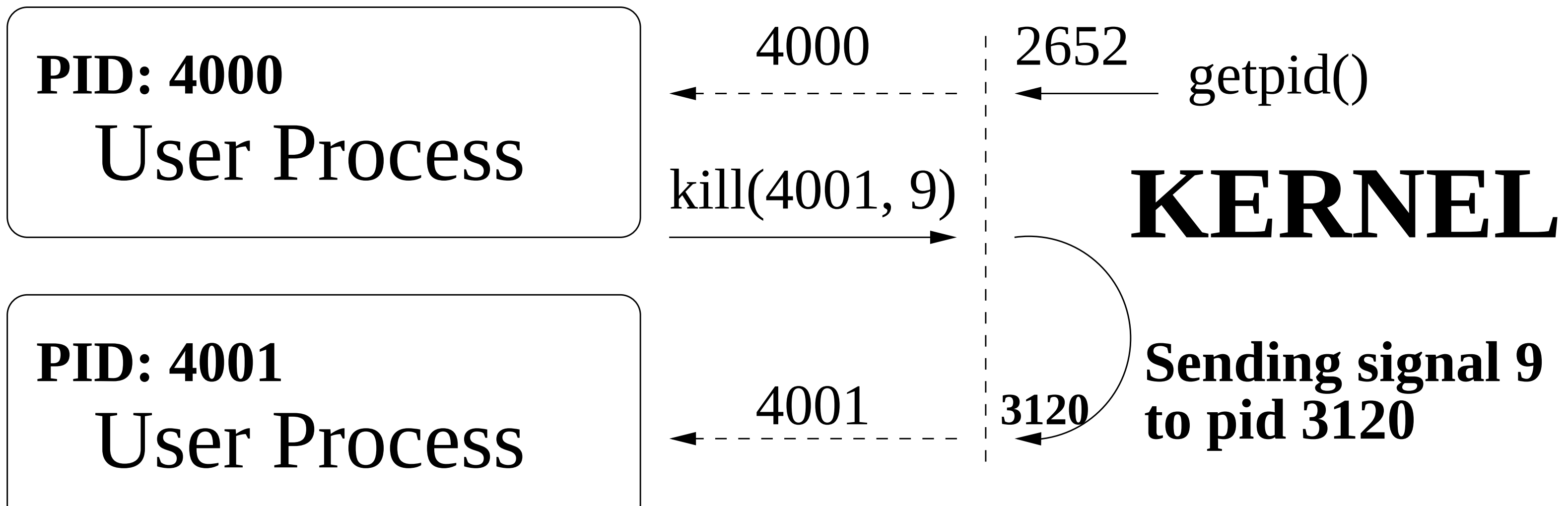
WHY PLUGINS?

- 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 modelling an external subsystem, and then creating a semantically equivalent construct at the time of restart.

A Simple DMTCP Plugin: Virtualizing the Process Id

- **PRINCIPLE:**
  The user sees only virtual pids; The kernel sees only real pids



Translation Table

Virt. PID	Real PID
4000	2652
4001	3120

Plugins for EDA: A Real-world Example

EDA is “Electronic Design Automation” (circuit design for chips).
Part of a four-year collaboration between DMTCP team and Intel:

> “Be Kind, Rewind — Checkpoint & Restore Capability for Improving Reliability of Large-scale Semiconductor Design”, I. Ljubuncic, R. Giri, A. Rozenfeld, and A. Goldis, IEEE HPEC-14, Sept., 2014.
> *(published solely by Intel co-authors)*

Fictional scenario with ball-park numbers (no particular vendor):

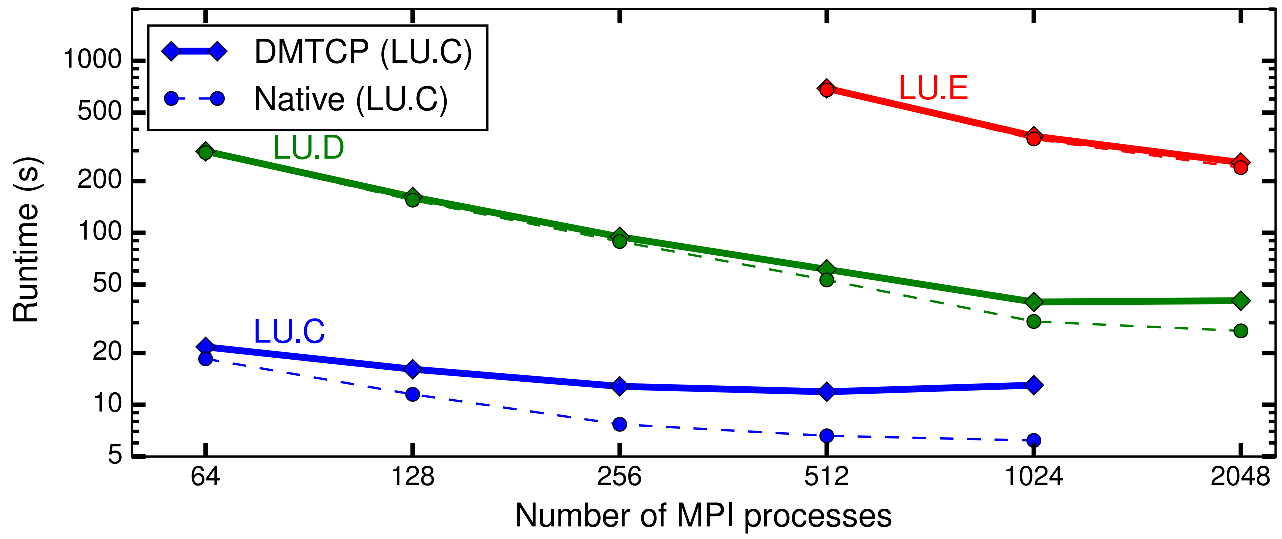
- Software circuit simulation: about 1 million times slowdown
- Hardware emulation at back-end: about 1 thousand times slowdown
- Cost of back-end hardware emulator: about $800,000
- **Use case A** (for a new CPU design): Boot Microsoft Windows overnight with emulator, and then test Microsoft Office.
- **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.*

Outline

Scalable checkpointing on Stampede at TACC

Using MVAPICH on Stampede in 2016:



For LU decomposition, the total size of data is constant. When the number of processes doubles, the size of a checkpoint image halves. So, the time halves. (This relationship breaks down at the level of 2K and 4K cores — apparently due to Lustre contention.)

Petascale checkpointing for two real-world applications

Using MVAPICH on Stampede in 2016:

- Petascale transparent, system-level checkpointing on Stampede:
  HPCG (**32,752 processes**, 652 s);
  NAMD (**16,368 processes**, 157 s)
  Compare w/ largest prev. publ. transp. checkpoint: 256 processes
- Stampede was the #10 supercomputer at the time of this experiment.
  1. *HPCG:* Combination of dense and sparse linear algebra intended to reflect real-world linear algebra applications
  2. *NAMD:* Molecular dynamics
- See: “System-level Scalable Checkpoint-Restart for Petascale Computing”, J. Cao, K. Arya, R. Garg, S. Matott, D.K. Panda, H. Subramoni, J. Vienne and G. Cooperman, 22nd IEEE ICPADS, 2016

Outline

Integration of DMTCP with Slurm

Checkpoint/restart in Slurm: current status and new developments,
M. Rodríguez-Pascual, J.A. Morigo, R. Mayo-García,
SLUG'16 (Slurm User's Groups),
`https://slurm.schedmd.com/SLUG16/ciemat-cr.pdf`
(Note: The authors are members of the CIEMAT institute, Madrid.)

This full integration of Slurm with DMTCP is now in beta testing as as a Slurm module. *(Additional beta test sites are welcome!)* This has resulted in several corner cases being fixed by Jiajun Cao and Manuel Rodríguez-Pascual. Many parts of the InfiniBand plugin for DMTCP transparent checkpointing have now been re-written for maintainability based on this feedback.

Batch Pools for Single-host Batch Queues (shared node)

QUESTION: If transparent checkpointing is now robust, why do we insist on a first-in, first-out protocol for batch queues?

ANSWER: Some first experiments have shown up to 20% better throughput by dynamically swapping jobs in and out to optimize co-scheduling.

Key Operating Regimes Supported:

Under-commitment of CPU cores: Given a large working set,
Set jobs: total threads (across all jobs) $<$ total CPU cores

Over-commitment of CPU cores: Given a small working set,
Set jobs: total threads (across all jobs) $>$ total CPU cores

Turn on Intel hyper-threading: Given small cache footprint,
Set jobs: total threads (across all jobs) $\gg$ total CPU cores

Support for OpenSHMEM

“Scalable System-level Transparent Checkpointing for OpenSHMEM”, Rohan Garg, Jérôme Vienne and Gene Cooperman, *OpenSHMEM and Related Technologies. Enhancing OpenSHMEM for Hybrid Environments — Third Workshop*, OpenSHMEM 2016, Baltimore, MD, USA, Aug. 2–4, 2016, Revised Selected Papers (OpenSHMEM’16), pp. 52–65, Lecture Notes in Computer Science, Volume 10007, Springer-Verlag

OpenhSHMEM: `http://www.openshmem.org/`
Standardized interface for shared memory view; supports PGAS/Partitioned Global Address Space, one-sided communication (similar to that of MPI and often implemented through RDMA: e.g., in InfiniBand), atomic operations, collective operations

Outline

Early Peek at Experimental Advances

- DMTCP support for statically linked executables (in progress) — Jay Kim
  (... and a first proof of principle to support Linux namespaces has separately been implemented; **Goal:** *Checkpoint Docker-based microservices and other Linux container-based technologies)*
- Transparent checkpointing for GPGPU computations (using NVIDIA GPUs) — Rohan Garg
  (first investigations only, but currently hopeful)
- Initial support for a simple case for Intel Omni-Path — Jiajun Cao
  (Omni-Path has better hardware support for MPI; some examples are a tagged architecture (MPI tags supported in hardware), and the registration of Omni-Path endpoints (think of an MPI rank) instead of InfiniBand queue pairs.)

Other Experimental Advances

- Full support for pty’s (pseudo-ttys) — Twinkle Jain
  (pty’s are often used to support potentially interactive features such as a terminal emulator, ssh to remote machine, etc.)
- Experimental support for combination of DMTCP transparent checkpointing with VeloC application-specific checkpointing — initial work by Rohan Garg
  (VeloC is a project of the DOE Exascale Initiative in the United States, led by Franck Cappello, Argonne National Laboratory.)

Questions?

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, Twinkle Jain, Samaneh Kazemi, Jay Kim, Gregory Kerr, Apoorve Mohan, Mark Mossberg, Manuel Rodríguez Pascual, Artem Y. Polyakov, Michael Rieker, Praveen S. Solanki, Ana-Maria Visan

QUESTIONS?

Supplementary Slides

SUPPLEMENTARY SLIDES

But How Does It Work?

Version 1:

1. Copy all of the process's virtual memory to a file.
   (It's easy under Linux:
   "`cat /proc/self/maps`" lists your memory regions.)

Version 2:

1. Make system calls to first discover the system state.
   "`ls /proc/self/fd`" to discover open files of the process.
   How much of file have we read?
   `current_offset = lseek(my_file_descriptor, 0, SEEK_CUR`
   And so on for other system state . . .
2. Copy all of the process's virtual memory to a file.

Version 3:

1. For distributed processes, drain "in-flight" network data into the memory of the process.
2. Make system calls to first discover the system state.
3. Copy all of the process's virtual memory to a file.

But How Does It Work? (details from operating systems)

- `dmtcp_launch ./a.out arg1 ...`

  $\searrow$

  `LD_PRELOAD=libdmtcp.so ./a.out arg1 ...`
- libdmtcp.so runs even before the user's `main` routine.
- libdmtcp.so:
  - libdmtcp.so defines a signal handler (for SIGUSR2, by default)
    (more about the signal handler later)
  - libdmtcp.so creates an extra thread: the *checkpoint thread*
  - The checkpoint thread connects to a DMTCP coordinator (or creates one if one does not exist yet).
  - The checkpoint thread then blocks, waiting for the DMTCP coordinator.

What Happens during Checkpoint? (details from operating systems)

1. The user (or program) tells the coordinator to execute a checkpoint.
2. The coordinator sends a ckpt message to the checkpoint thread.
3. The checkpoint thread sends a signal (SIGUSR2) to each user thread.
4. The user thread enters the signal handler defined by libdmtcp.so, and then it blocks there.

   (Remember the SIGUSR2 handler we spoke about earlier?)
5. Now the checkpoint thread can copy all of user memory to a checkpoint image file, while the user threads are blocked.

Anatomy of a Plugin

Plugins support three essential properties:

Wrapper functions: Change the behavior of a system call or call to a library function (X11, OpenGL, MPI, . . .), by placing a wrapper function around it.

Event hooks: When it's time for checkpoint, resume, restart, or another special event, call a "hook function" within the plugin code.

Publish/subscribe through the central DMTCP coordinator: Since DMTCP can checkpoint multiple processes (even across many hosts), let the plugins within each process share information at the time of restart: publish/subscribe database with key-value pairs.

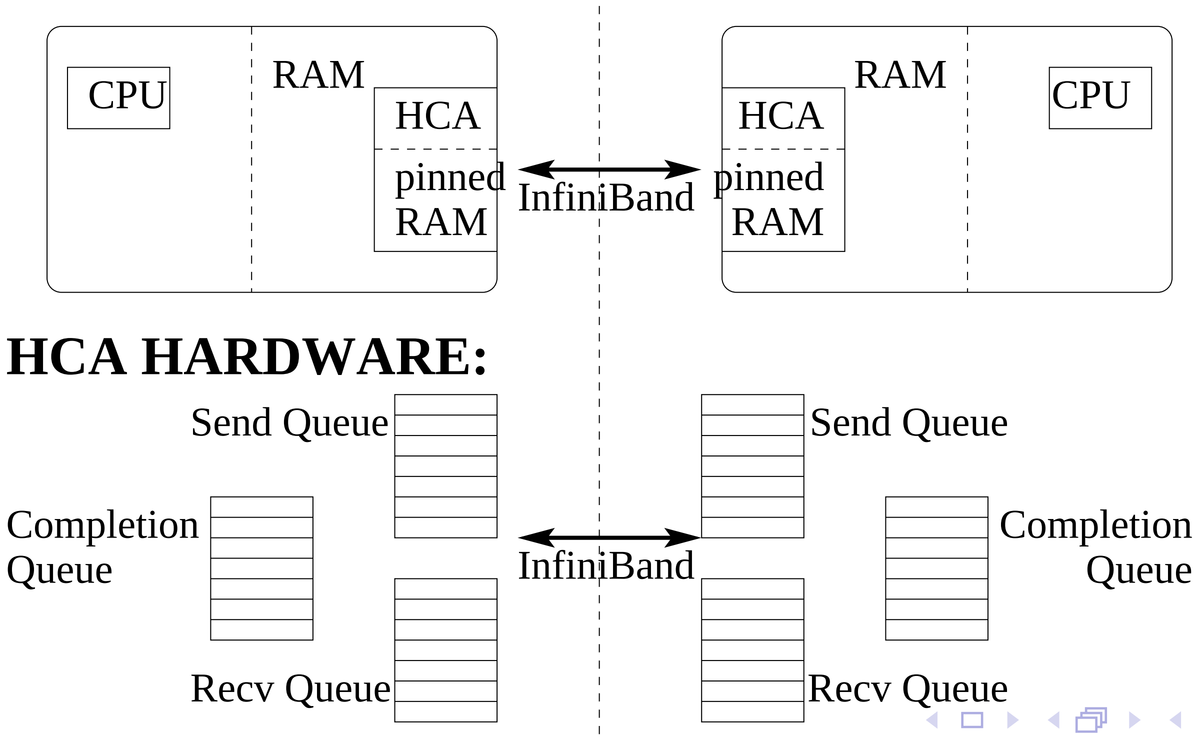
InfiniBand Plugin

Checkpoint while the network is running! *(Older implementations tore down the network, checkpointed, and then re-built the network.)*

Design the plugin once for the API, not once for each vendor/driver!

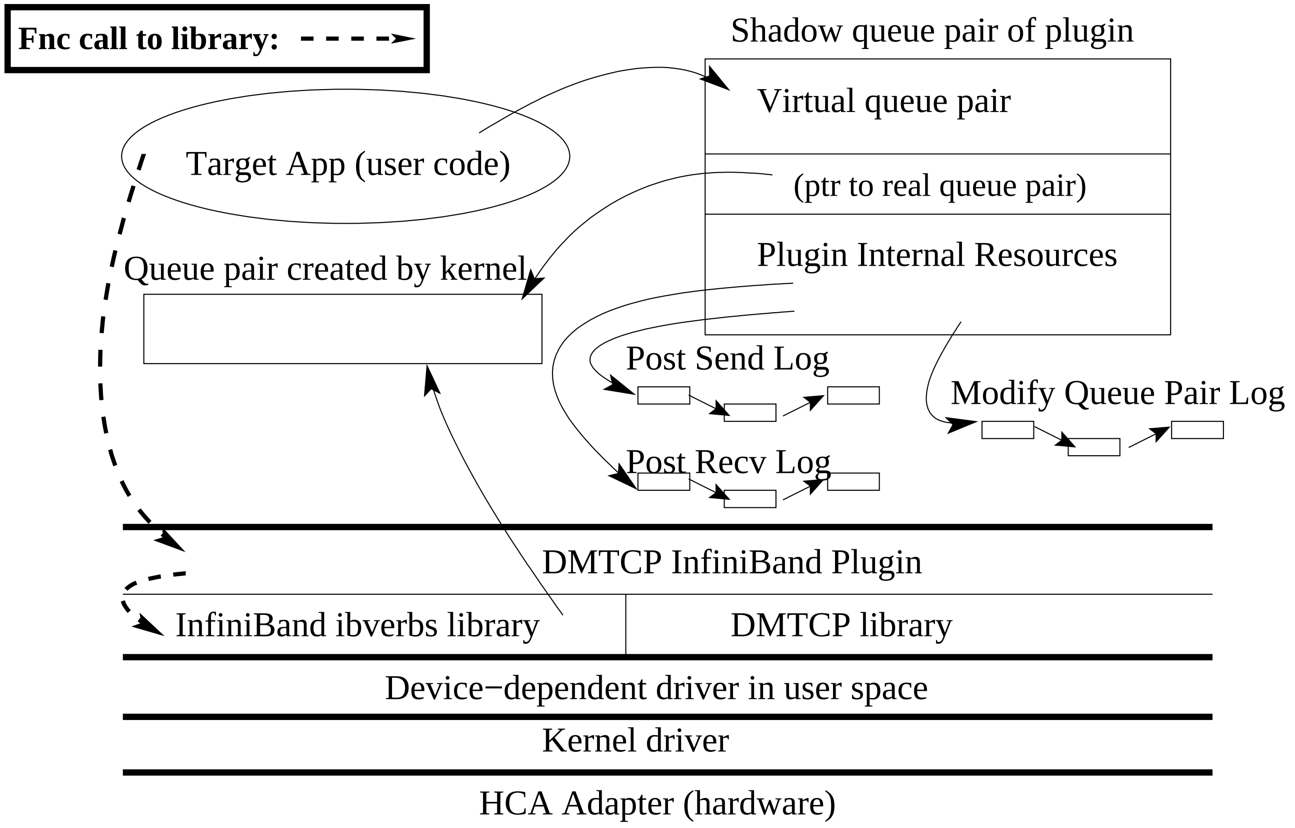
socket plugin: `ipc/socket`; *InfiniBand plugin:* `infiniband`

- InfiniBand uses RDMA (Remote Direct Memory Access).
  InfiniBand plugin is a model for newer, future RDMA-type APIs.
- Virtualize the *send queue*, *receive queue*, and *completion queue*.



DMTCP and InfiniBand

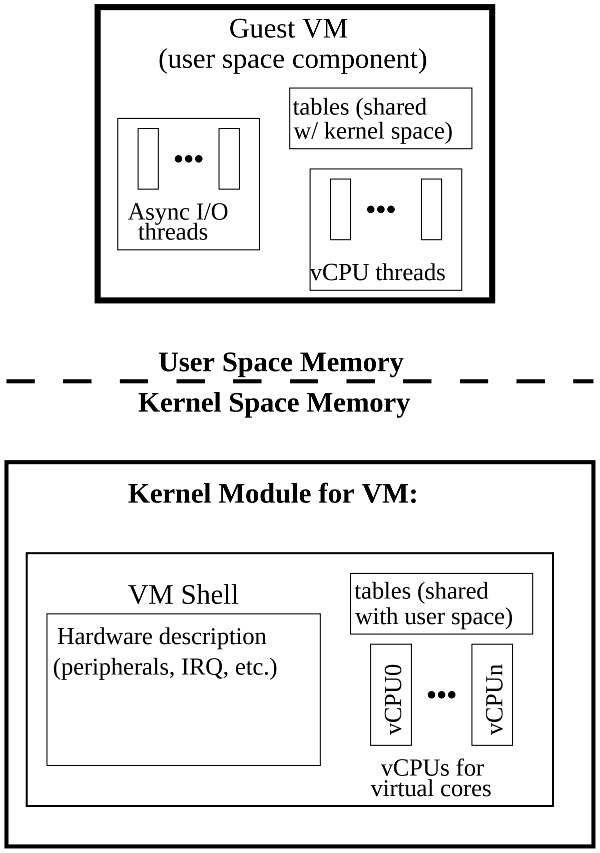
- *ISSUES:* At restart time, totally different ids and queue pair ids.
- *Solution:* Drain the completion queue and save in memory.
  On restart, virtualize the completion queue:
  - Virtualized queue returns drained completions before returning completions from the hardware.

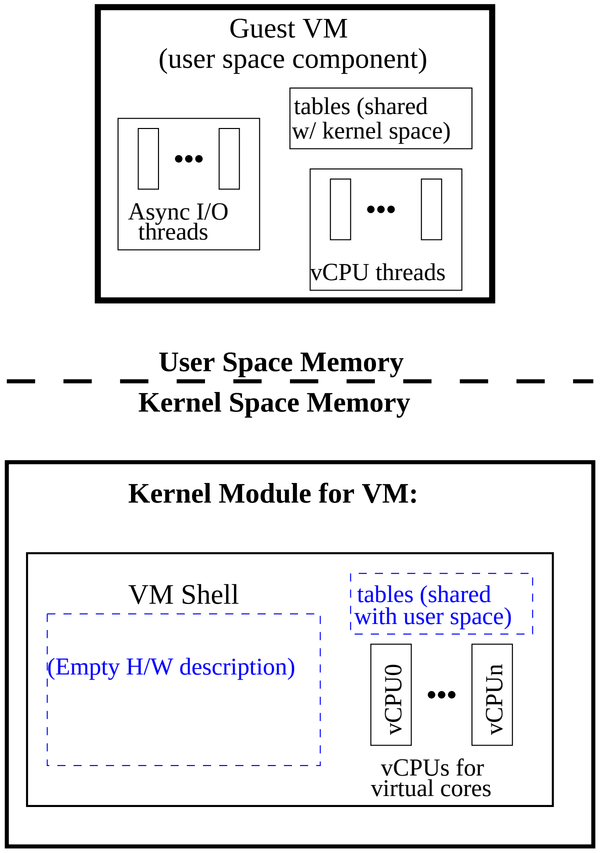


See: *Transparent Checkpoint-Restart over InfiniBand*, HPDC-14, Cao, Kerr, Arya, Cooperman

KVM Plugin: Checkpoint a Virtual Machine

- *Issue:* KVM acts as a hypervisor that will launch guest virtual machines. How to “re-launch” a previously checkpointed VM?
- *Solution:* Virtualize the KVM API for a guest (QEMU) virtual machine





Tun Plugin: Checkpoint a Network of Virtual Machines

- *Issue:* Current virtual machine snapshots cannot also save the state of the network. (Networking virtual machines requires the Linux Tun/Tap kernel module.)
- *Solution:* Virtualize the KVM API for a guest (QEMU) virtual machine
  NEXT: Virtualize the Tun network.
  Write a DMTCP plugin to save the state of the "Tun" network between virtual machines on different physical nodes.

  "Checkpoint-Restart for a Network of Virtual Machines",
  Rohan Garg, Komal Sodha, Zhengping Jin and and Gene Cooperman,
  Proc. of 2013 IEEE Cluster Computing
  `http://www.ccs.neu.edu/home/gene/papers/cluster13.pdf`

OpenGL Plugin: Checkpoint 3-D Graphics

- Usually a virtual machine cannot take a snapshot of 3-D graphics (cannot snapshot OpenGL applications). This is because the 3-D graphics object are saved in the graphics hardware.
- *Issue:* Same problem as we saw with InfiniBand hardware.
  What is the solution this time?
- *Solution:* Record, compress, and replay the commands.
  Virtualize the graphics objects in the graphics hardware accelerator.
- "Transparent Checkpoint-Restart for Hardware-Accelerated 3D Graphics",
  Samaneh Kazemi Nafchi, Rohan Garg, and Gene Cooperman
  `http://arxiv.org/abs/1312.6650`

EXAMPLE: Plugin Event

```
void dmtcp_event_hook(DmtcpEvent_t event,
                      DmtcpEventData_t *data)
{
  switch (event) {
  case DMTCP_EVENT_WRITE_CKPT:
    printf("\n*** Checkpointing. ***\n"); break;
  case DMTCP_EVENT_RESUME:
    printf("*** Resume: has checkpointed. ***\n"); break;
  case DMTCP_EVENT_RESTART:
    printf("*** Restarted. ***\n"); break;
  ...
  default: break;
  }
  DMTCP_NEXT_EVENT_HOOK(event, data);
}
```

EXAMPLE: Plugin Wrapper Function

```
unsigned int sleep(unsigned int seconds)
{ /* Same type signature as sleep */
  static unsigned int (*next_fnc)() = NULL;
  struct timeval oldtv, tv;
  gettimeofday(&oldtv, NULL);
  time_t secs = val.tv_sec;
  printf("sleep1: "); print_time(); printf(" ... ");
  unsigned int result = NEXT_FNC(sleep)(seconds);
  gettimeofday(&tv, NULL);
  printf("Time elapsed:  %f\n",
          (1e6*(val.tv_sec-oldval.tv_sec)
           + 1.0*(val.tv_usec-oldval.tv_usec)) / 1e6);
  print_time(); printf("\n");

  return result;
}
```

Some Example Strategies for Writing Plugins

- *Virtualization of ids:* see pid virtualization — ≈ 50 lines of code
- *Virtualization of protocols (example 1):* virtualization of ssh daemon (sshd) — ≈ 1000 lines of code
- *Virtualization of protocols (example 2):* virtualization of network of virtual machines — ≈ 750 lines of code (KVM/QEMU) and ≈ 350 lines of code (Tun/Tap network)
- *Shadow device driver:* transparent checkpointing over InfiniBand — ≈ 3,600 lines of code
- *Record-Replay with pruning:* transparent checkpointing of 3-D graphics in OpenGL for programmable GPUs — ≈ 4,500 lines of code
- *Record state of O/S subsystem and CPU:* checkpointing of ptrace system call for GDB, etc. — ≈ 1,000 lines of code (includes checkpointing x86 eflags register, trap flag: CPU single-stepping)