Applications of DMTCP for Checkpointing

Gene Cooperman
gene@ccs.neu.edu

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

May 26, 2016

*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).

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:

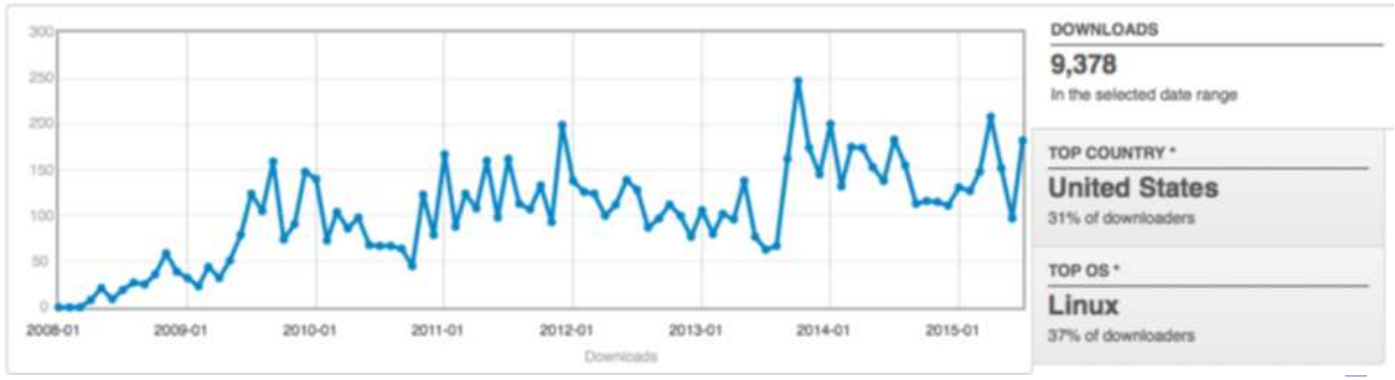
```
dmtcp_launch ./a.out
dmtcp_command --checkpoint
dmtcp_restart ckpt_myapp_*.dmtcp
```

- 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:*



What is Checkpointing?

Checkpointing is the action of saving the state of a running process to a checkpoint image file.

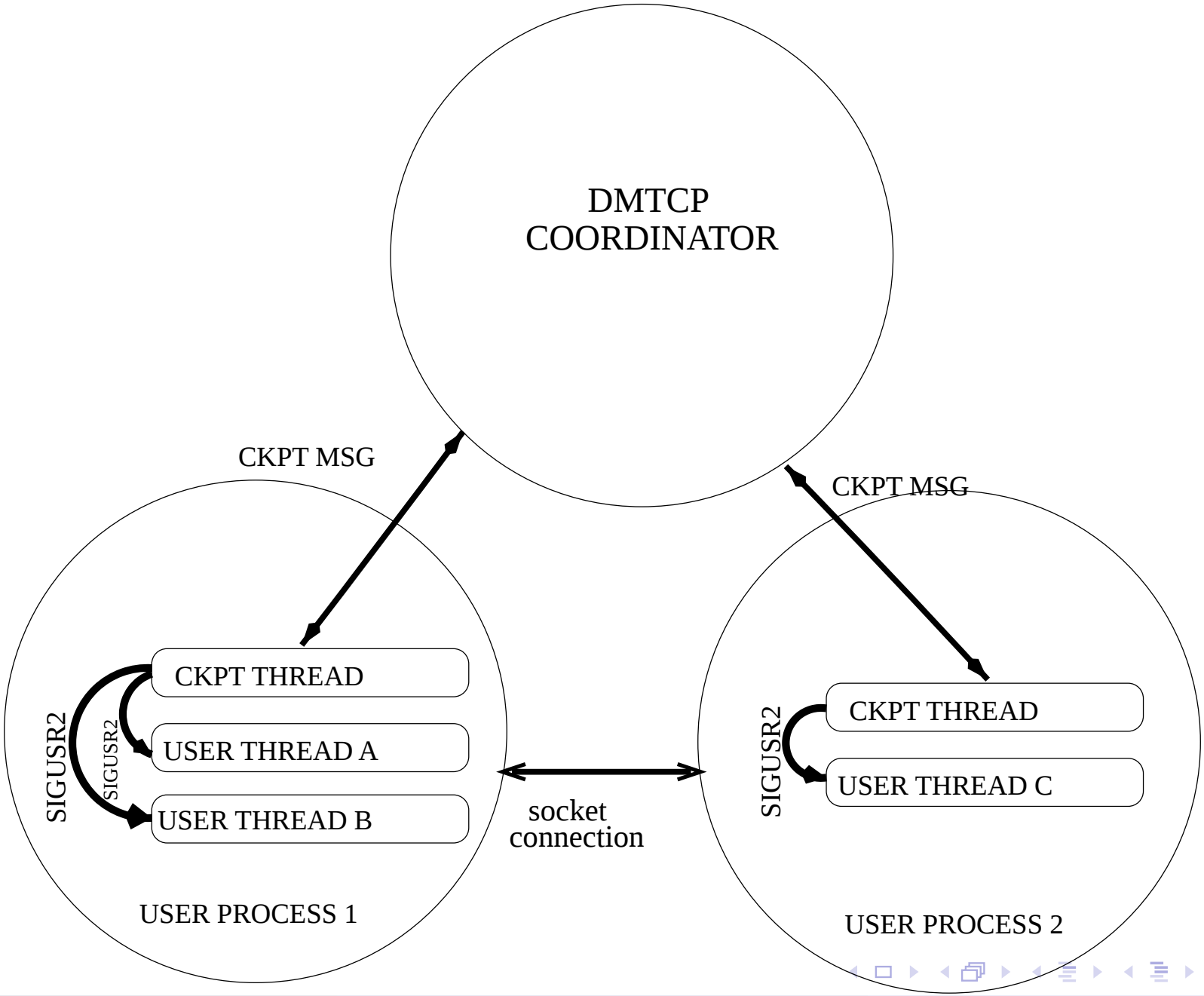
Checkpointing supports several other features for free!

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



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 ...`

  ↘

  `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.

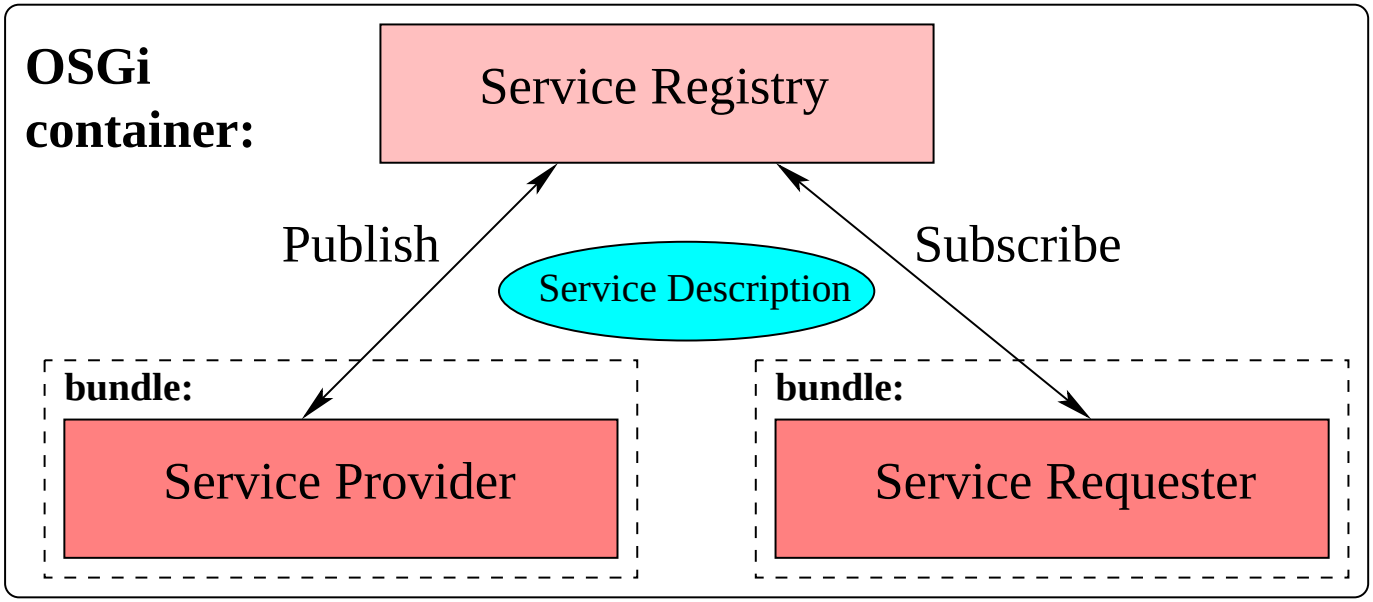
What Happens during Checkpoint? (details from operating systems)

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.

Outline

OSGi Model

OSGi: Open Services Gateway initiative

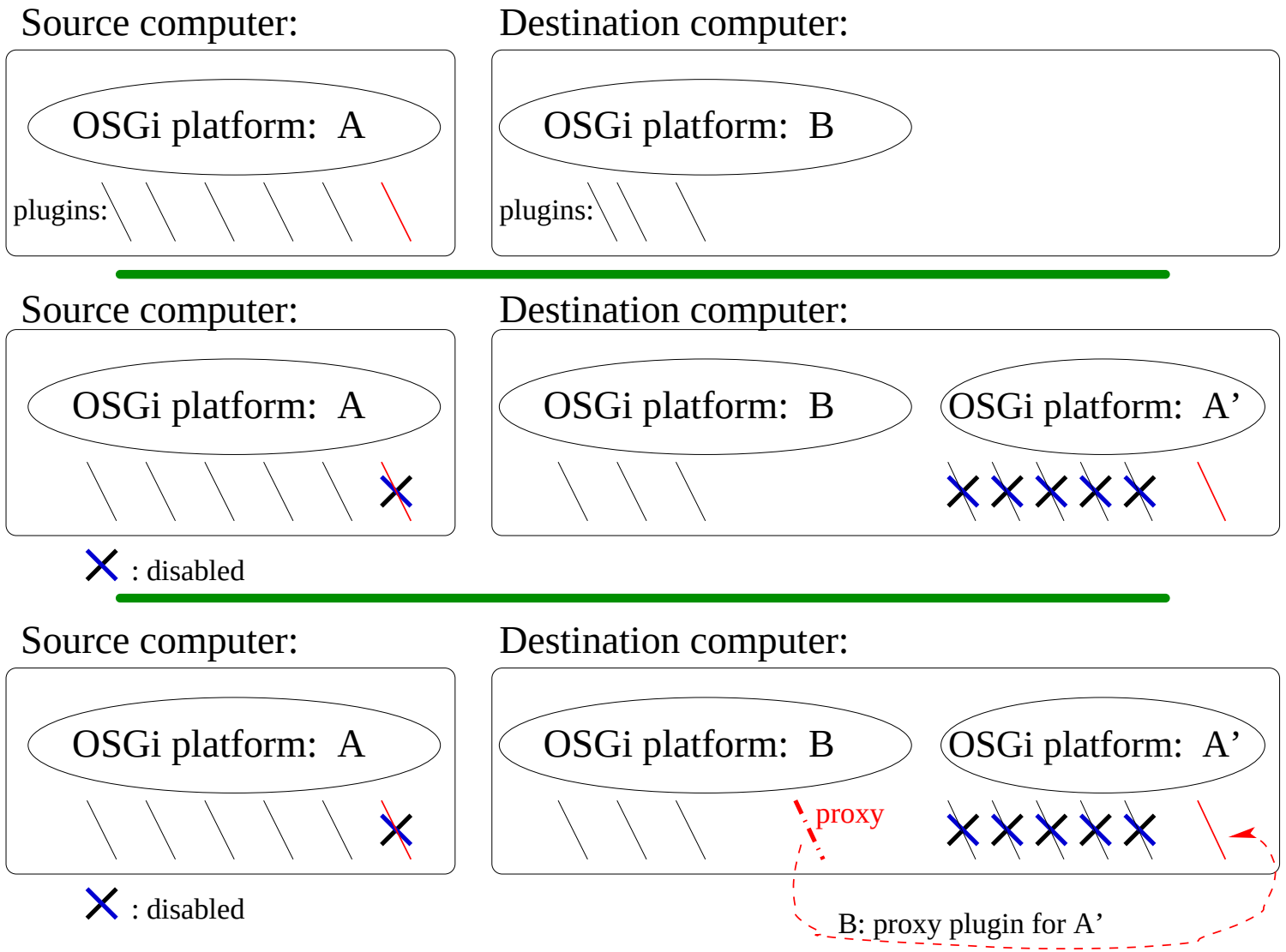


Simplified diagram:

bundles (plugins):

Migrating an OSGi Plugin between Computers

VISION: I. checkpoint A; **II.** restart as A' (a copy of A) on destination;
III. disable unused plugins; **IV.** add a proxy so that B can use plugins of A'



Outline

DMTCP Plugins

WHY PLUGINS?

- Processes must talk with the rest of the world!
  (This is the same issue as OSGi plugins for IoT.)
- *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.

DMTCP Plugins (Demo: part 1)

```
> SLEEP1% ls
> Makefile  README  sleep1.c
> SLEEP1% vi sleep1.c
> SLEEP1% make -n
> gcc -fPIC -I../../../include -c -o sleep1.o sleep1.c
> gcc -shared -fPIC -o libdmtcp_sleep1.so sleep1.o
> SLEEP1% make && ls
> libdmtcp_sleep1.so  Makefile  README  sleep1.o sleep1.c
> SLEEP1% make -n check
> ../../../bin/dmtcp_launch --interval 5 \
>  --with-plugin $PWD/libdmtcp_sleep1.so ../../../test/dmtcp1
```

DMTCP Plugins (Demo: part 2)

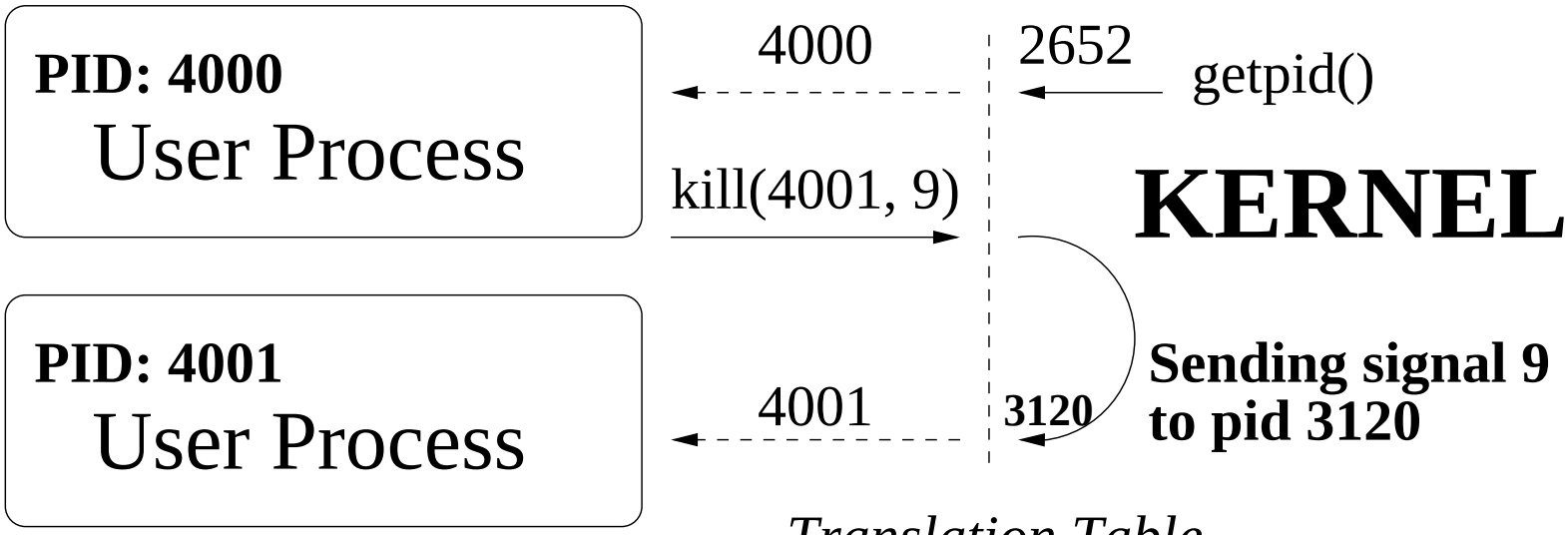
```
> SLEEP1% ../../../test/dmtcp1
>   1   2   3   4 ^C
> SLEEP1% ../../../bin/dmtcp_launch --interval 5 \
>  --with-plugin $PWD/libdmtcp_sleep1.so ../../../test/dmtcp1
>   1 sleep1: 1464197122 987160 ... 1464197124 987252
>   2 sleep1: 1464197124 987270 ... 1464197126 987355
>   3 sleep1: 1464197126 987370 ...
> *** The plugin sleep1.c is being called before checkpointing
> *** The plugin sleep1.c has now been checkpointed. ***
> 1464197128 400509
>   4 sleep1: 1464197128 400522 ... 1464197130 400614
>   5 ^C
```

DMTCP Plugins (Demo: part 3)

```
vi sleep1.c
> void print_time() {
>   struct timeval val;
>   gettimeofday(&val, NULL);
>   printf("%ld %ld", (long)val.tv_sec, (long)val.tv_usec); }
>
> unsigned int sleep(unsigned int seconds) {
>   printf("sleep1: "); print_time(); printf(" ... ");
>   unsigned int result = NEXT_FNC(sleep)(seconds);
>   print_time(); printf("\n");
>   return result; }
>
> static void checkpoint() {
>   printf("\n*** The plugin %s is being called before checkpo
>          __FILE__);
> }
>
```

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

Other Applications Of DMTCP

FROM: `http://dmtcp.sourceforge.net/publications.html`

- Debugging a simulated CPU model at Intel Corporation:
  "Be Kind, Rewind — Checkpoint & Restore Capability for Improving Reliability of Large-scale Semiconductor Design", Ljubuncic et al., HPEC-2014
- "Direct Inference of Protein—DNA Interactions using Compressed Sensing Methods", AlQuraishi et al.,
  Proc. of National Academy of Sciences (PNAS), 2011
- An online site for interactive theorem proving
- Live migration in support of a green, energy saving cloud
- Software model checking
- Energy efficient processing of big data
- . . . **(50 refereed papers by others in the published literature)**

Questions?

QUESTIONS?

SUPPLEMENTARY SLIDES

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.

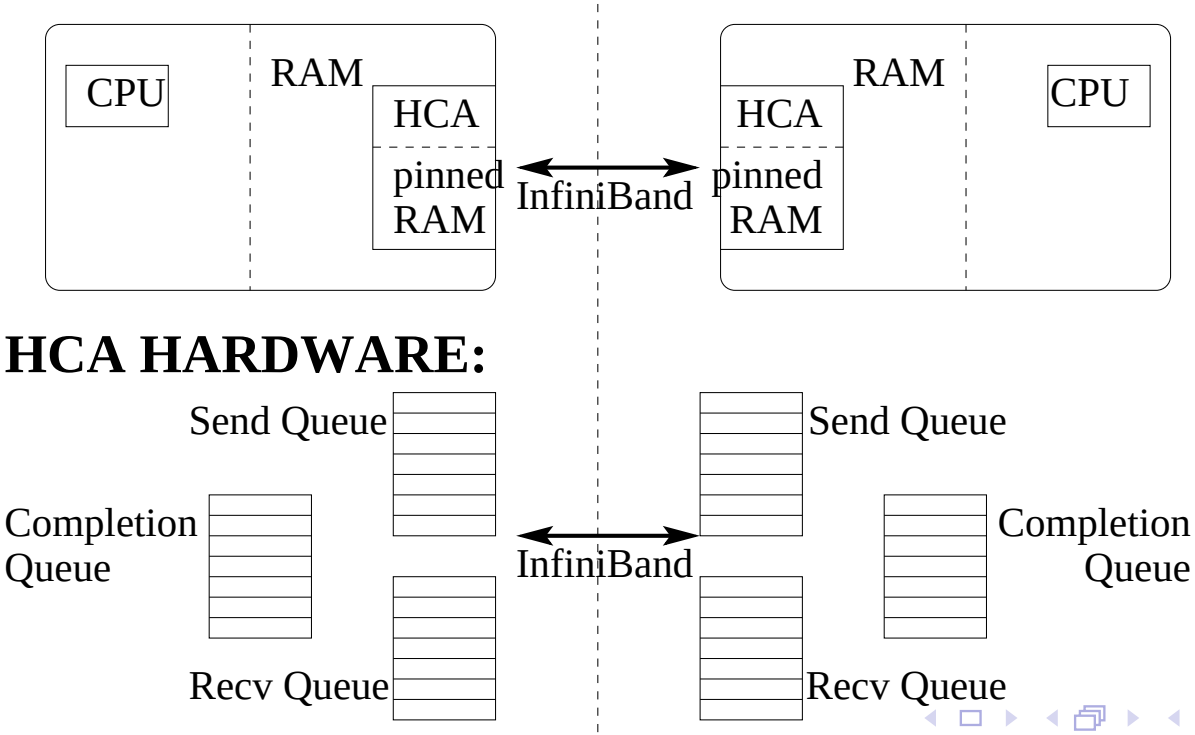
InfiniBand Plugin

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!

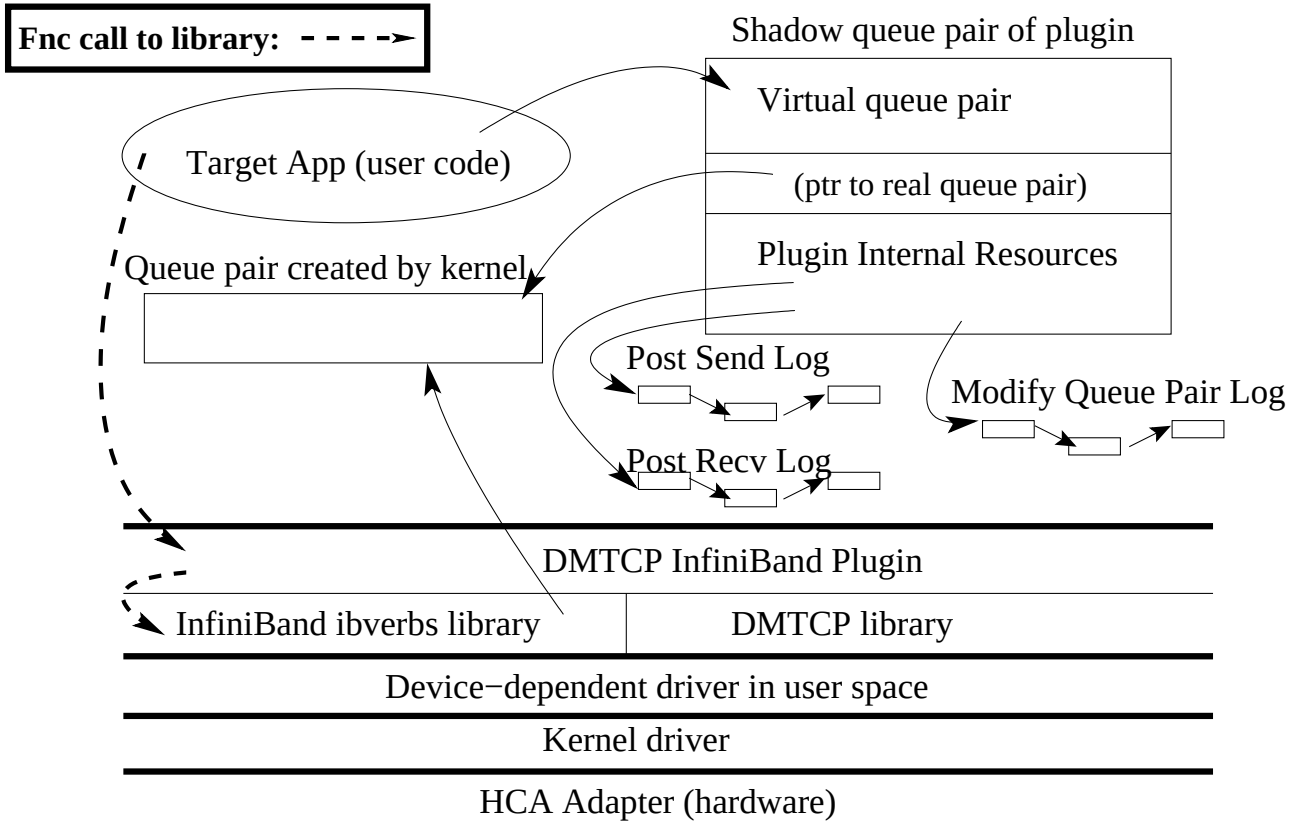
socket plugin: `ipc/socket`; *InfiniBand plugin:* `infiniband`

- 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

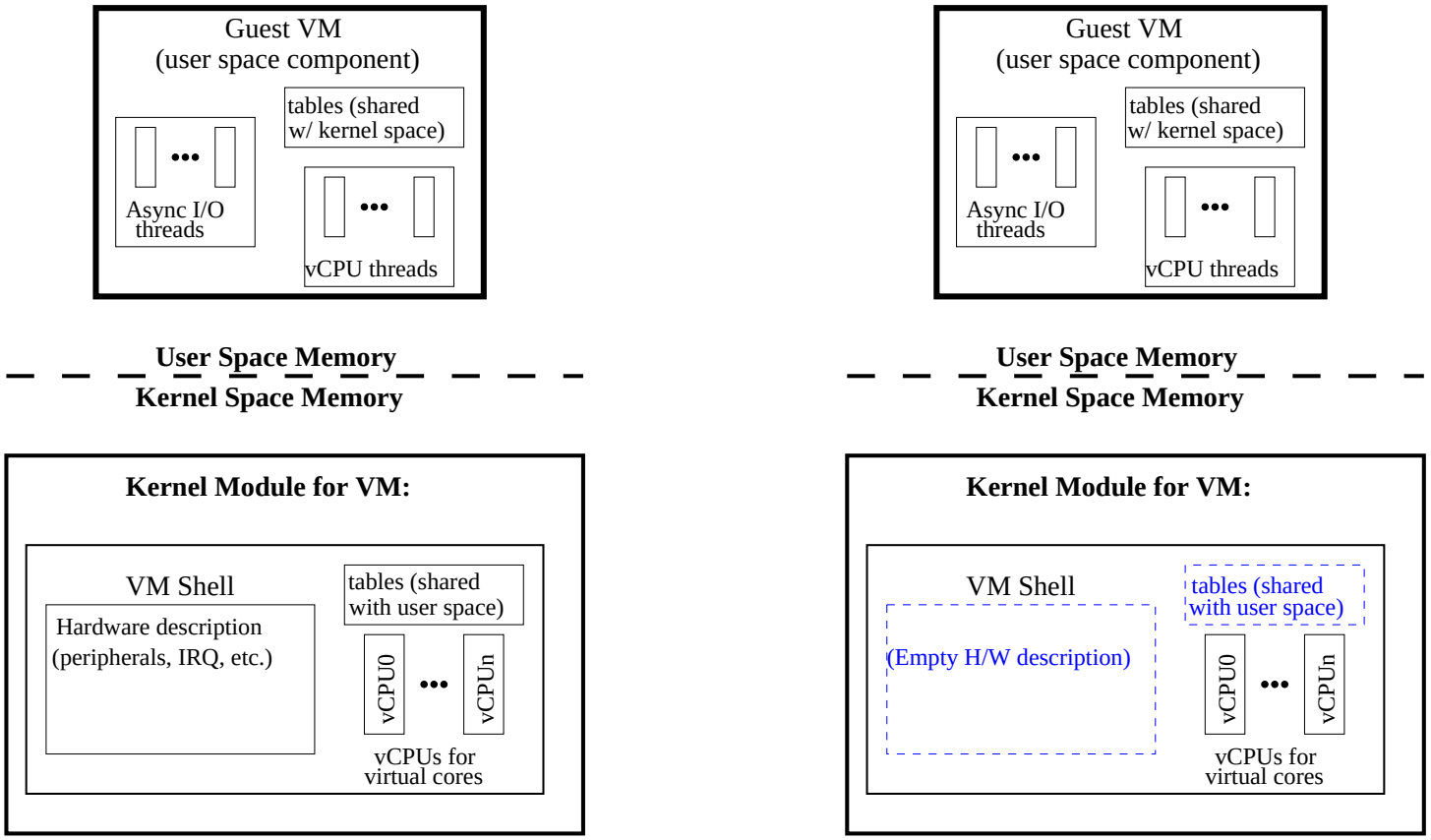
- *ISSUES:* At restart time, totally different ids and queue pair ids.
- *Solution:* Drain the completion queue and save in memory.
  On restart, virtualize the completion queue:
  - Virtualized queue returns drained completions before returning completions from the hardware.



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)