# A Brief History of Checkpointing and Its Applications

Gene Cooperman
gene@ccs.neu.edu

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

July 7, 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:

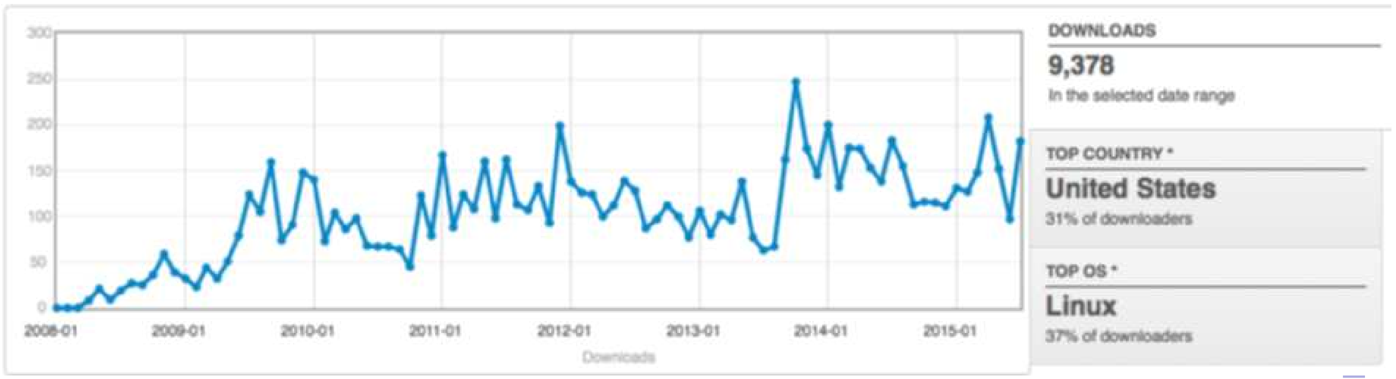
```
dmtcp_launch ./myapp
dmtcp_command --checkpoint
dmtcp_restart ckpt_myapp_*.dmtcp
```

- 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
9,378
In the selected date range

TOP COUNTRY *
United States
31% of downloaders

TOP OS *
Linux
37% of downloaders

## What is Checkpointing?

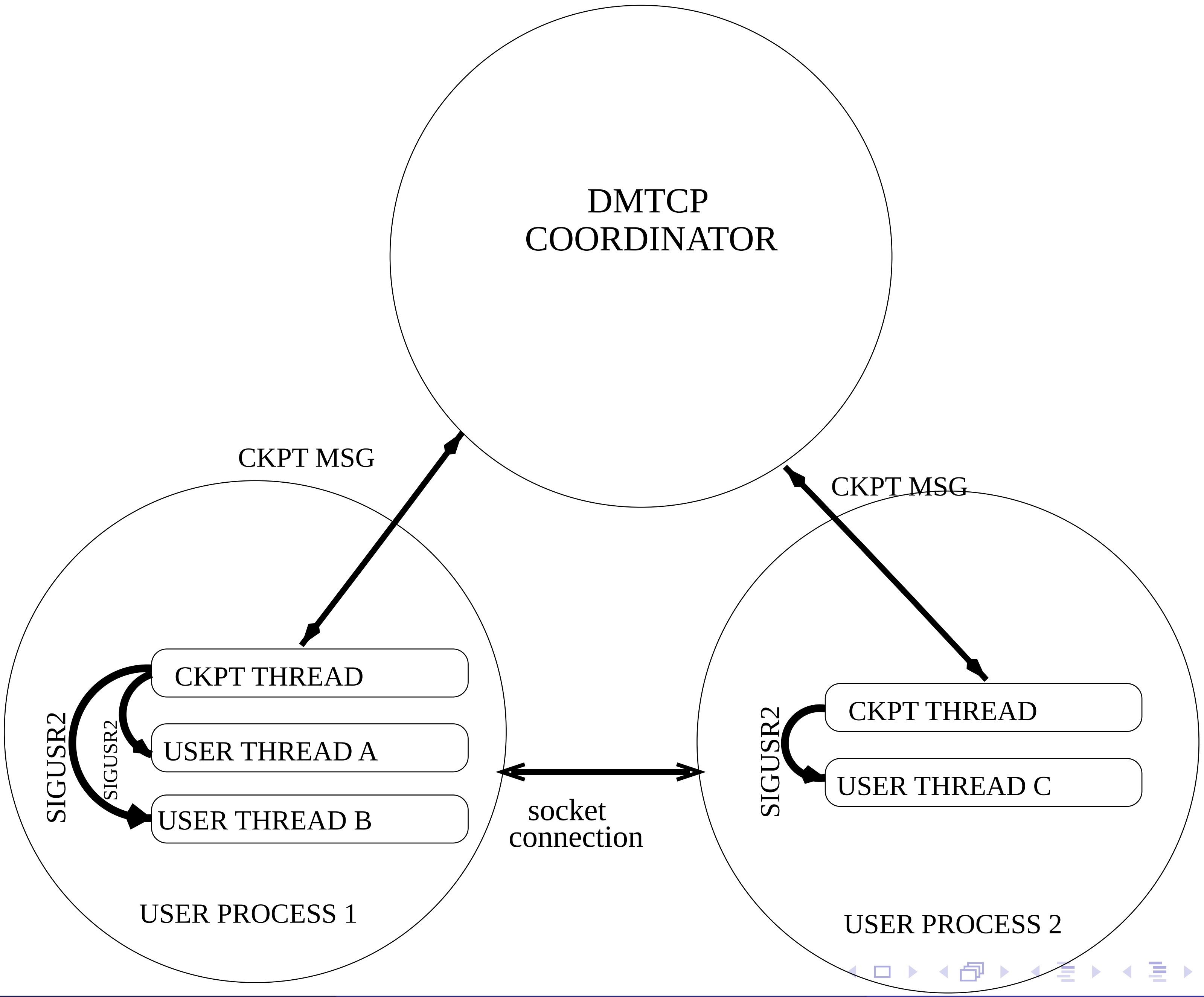
*Checkpointing* is the action of saving the state of a running process to a checkpoint image file.
**Checkpointing supports several other features for free!**

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.

## 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.

**Scalability was recently demonstrated by checkpointing 24,000 processes (HPCG/MPI computation on Stampede/TACC supercomputer)!**

## Outline

## Problem: You Can't Checkpoint the World!

**Problems often encountered:**

- NSCD daemon, license server, etc.: caching remote information, remote permissions
- External server on Internet
- 3D graphics state built on top of hardware state in the GPU
- Network data in flight (residing in network switch)
- Large database (too large to checkpoint)
- Migrating to new hosts (new network addresses, new pathnames)

## Solution 1: Some Applications Fix Themselves

- TCP is supposed to provide reliable, connection-oriented communication.
- But in the real world, connections “break”, and robust applications must know how to re-establish a connection with an external server.
- On restart after checkpoint, DMTCP simulates a broken connection to the external server.
- The robust application perceives this as a broken connection, and it re-establishes the server connection after restart.

## Solution 2: Application-specific Solutions

A widely used example of an application-specific solution is the usage of a checkpoint-restart service within an MPI library. *(MPI is the most widely used standard for message-passing, for communication in parallel applications over distributed nodes.)*

- Just prior to checkpoint, the checkpoint-restart service tears down the network.
- The job of checkpointing each single process is then delegated to a single-process checkpoint package (typically, BLCR).
- The network is then re-initialized after checkpoint.

Hence, upon restart after failure, the individual processes are restarted first, without any network connection, and then the MPI checkpoint-restart service re-initializes the original network with the original topology.

## Solution 2: Problem with Application-specific Checkpointing

One developer is in charge of a subsystem with state that needs to be checkpointed.

A second developer is in charge of the application-specific checkpointing routine to save the state of all subsystems.

The first developer then updates the subsystem state in a manner that is not well-understood by the second developer.

**Anecdotally, this software engineering issue is often the root cause, when an application-specific routine "used to work", and then it stops working.**

## Outline

## Solution 3: DMTCP Plugins

**WHY PLUGINS?**

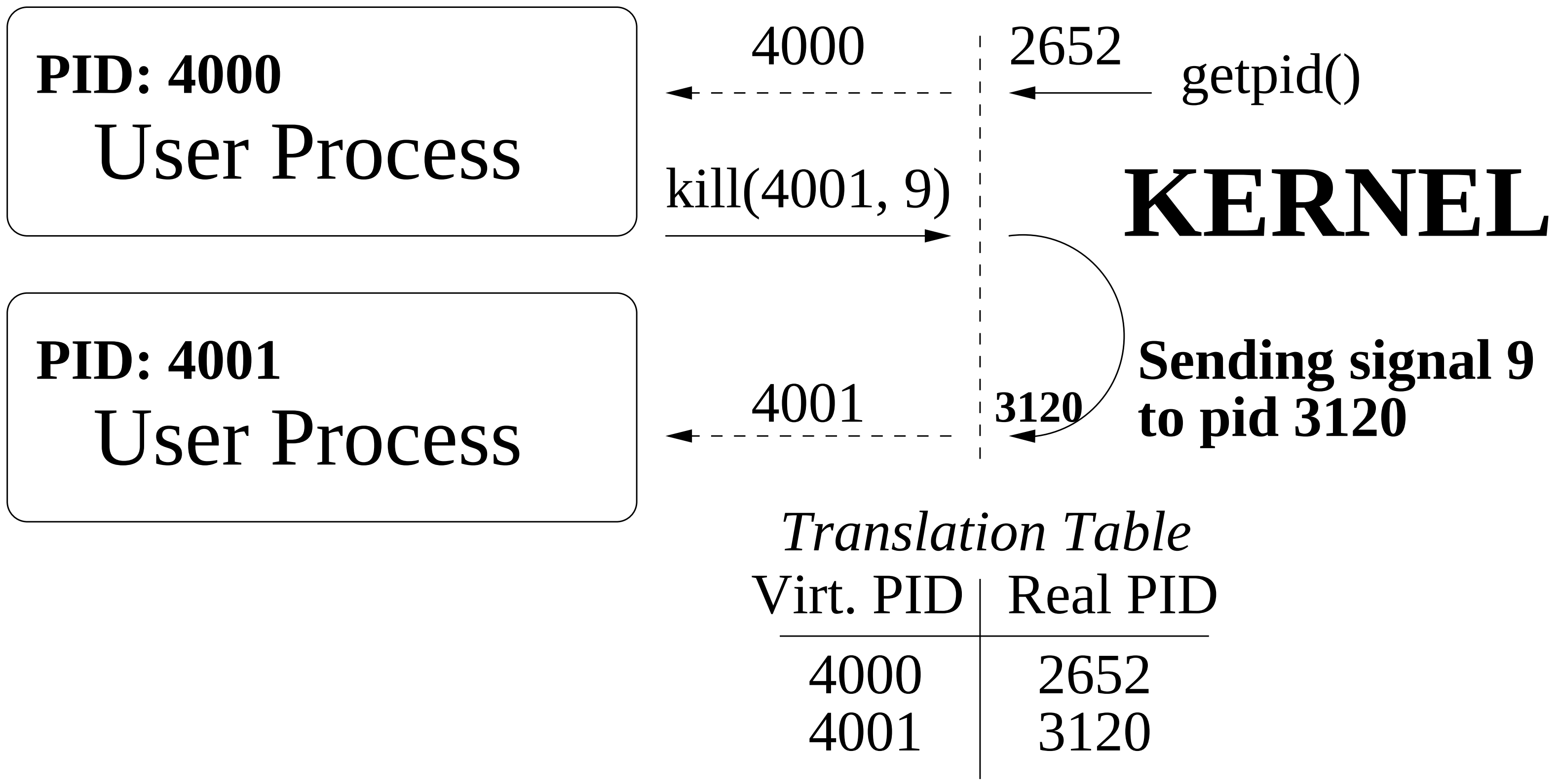
- Processes must talk with the rest of the world!
- *Process virtualization:* virtualize the connections to the rest of the world

**In short, a plugin is responsible for modelling an external subsystem, and then creating a semantically equivalent construct at the time of restart.**

**SLIDES-BASED DEMO LATER IN TALK (time permitting)**

## 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 |

## 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 checkp
>          __FILE__);
> }
>
```

## Outline

## Other Applications Of DMTCP

- Early origins of checkpointing in Condor and MOSIX. EXAMPLE:
  1. Condor starts a job on an unused computer (e.g., on a student lab computer, when unoccupied)
  2. A student sits down and uses that lab computer.
  3. Condor migrates job to a new computer, and runs a small "stub" program to retain access to any temporary files used on secretary's computer.
- Many early checkpointing systems *used to be* at `checkpointing.org`: `http://web.archive.org/web/20140517053408/http://checkpoin`
- Some checkpointing packages that received wider usage:
  1. BLCR (Berkeley Laboratory Checkpoint-Restart): single-host checkpointing implemented via a kernel module: foundation for many checkpoint-restart services for MPI for parallel computing.
  2. Cryopid-2: Uses ptrace syscall. Modest, but easy-to-use for simple apps.
  3. CRIU (Checkpoint-Restart In User-space): Uses extended proc filesystem to expose kernel internals; Operates in user space, but may require admin privilege to access full proc filesystem interface (e.g., security concerns).
  4. DMTCP: Handles distributed processes entirely in user-space; no kernel modifications; no need for admin privileges

## 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?

THANKS TO THE MANY STUDENTS WHO HAVE CONTRIBUTED TO DMTCP OVER THE YEARS:

Jason Ansel, Kapil Arya, Alex Brick, Jiajun Cao, Tyler Denniston, Xin Dong, William Enright, Rohan Garg, Samaneh Kazemi, Gregory Kerr, Apoorve Mohan, Mark Mossberg, Artem Y. Polyakov, Michael Rieker, Praveen S. Solanki, Ana-Maria Visan

**QUESTIONS?**

## Supplementary Slides

SUPPLEMENTARY SLIDES

## Principles (cont.)

- Entirely based on user-space: If one checkpoints on an older O/S kernel, one can restart on a newer O/S kernel. *(But if one checkpoints on a newer kernel, the library may use a newer kernel system call that doesn't exist in the older kernel.)*
- Debugging with GDB on restart is possible. (See DMTCP FAQ.)
- Currently supports Intel and ARM. (Currently, 138 lines of assembly.)
- Checkpoints can be invoked: periodically; under program control; or under external control.
- What is the time to checkpoint? (It's mostly due to the time to write to stable storage (e.g., disk).)
- It is possible to omit saving some process memory ("cutouts").
- What is the run-time overhead of DMTCP? (It's too small to measure on real-world programs. The DMTCP overhead is entirely due to "thin wrappers" around certain system calls.)

## 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.

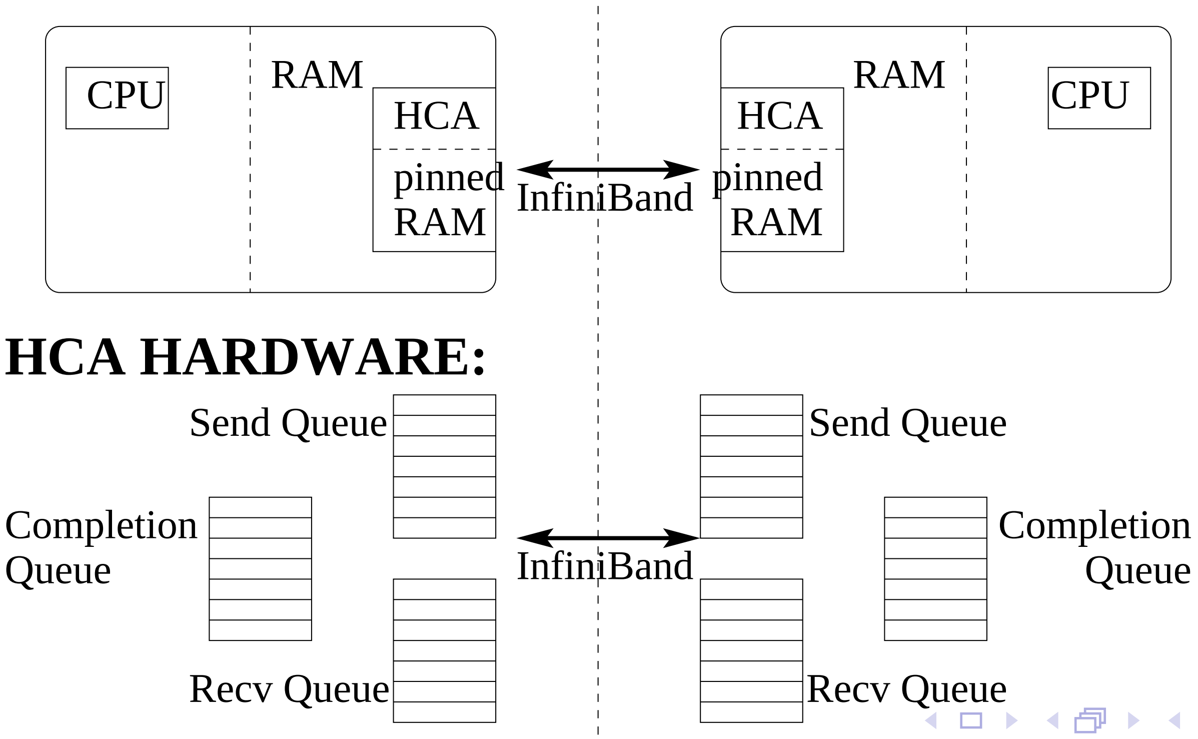
Publish/subscribe through the central DMTCP coordinator: Since DMTCP can checkpoint multiple processes (even across many hosts), let the plugins within each process share information at the time of restart: publish/subscribe database with key-value pairs.

## InfiniBand Plugin

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

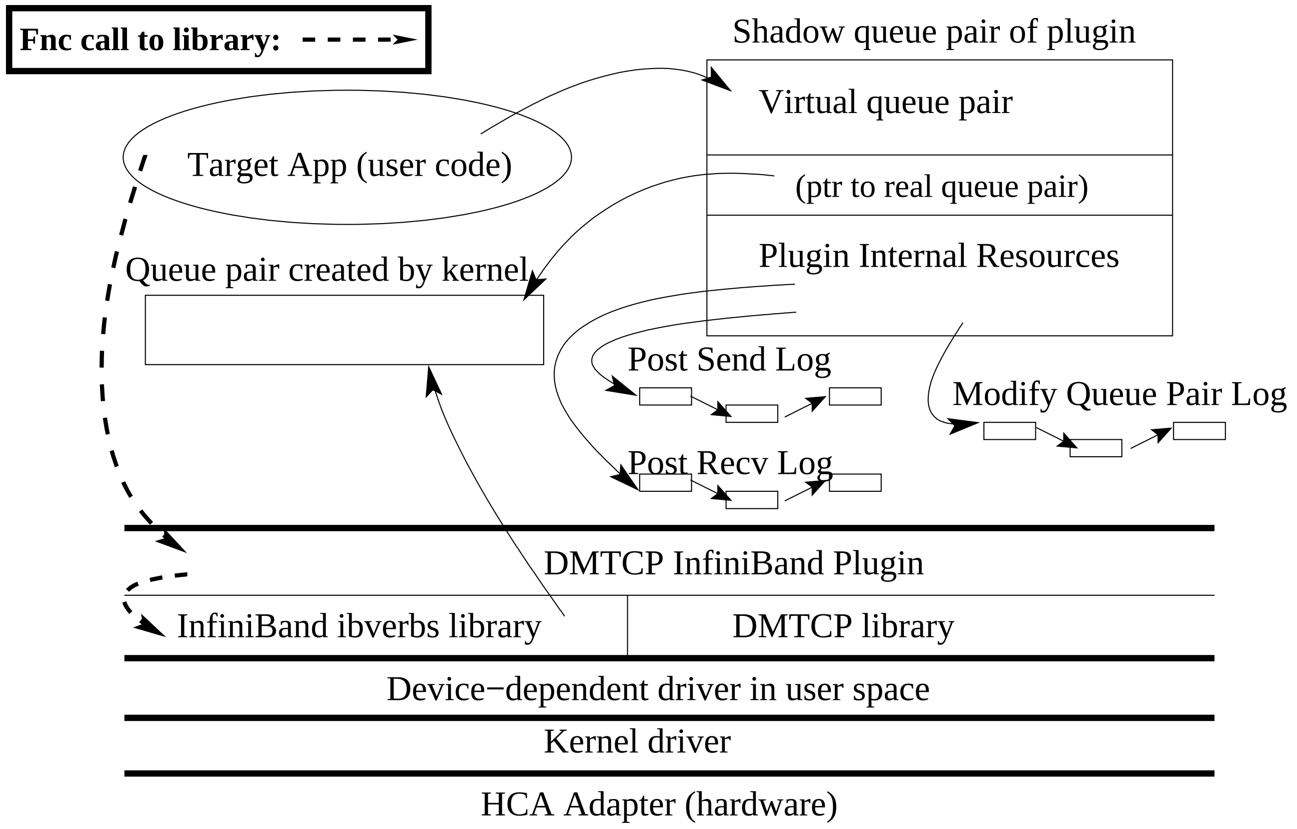
**Design the plugin once for the API, not once for each vendor/driver!**
*socket plugin:* `ipc/socket`; *InfiniBand plugin:* `infiniband`

- 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

## 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)

## Speculation on Potential DMTCP Extensions

**CAVEAT! None of this has been implemented.**

> *Based on a knowledge of the internals of DMTCP, we believe that each of these extensions to DMTCP are possible. But only an example implementation can verify this.*

Live Migration: Classical algorithm; works best when most of memory only changes slowly.

Partial Restart: Restart only some of the processes, and re-connect them to existing processes.

## Speculation on Potential DMTCP Extensions (cont.)

Merging DMTCP Computations: One DMTCP coordinator takes responsibility on restart for two separate DMTCP computations (or similarly, split two DMTCP computations).

Graphics "Desktop": DMTCP has been used with VNC to checkpoint a graphics desktop. (Useful for graphics instrumentation panel?)

Migration across CPUs: Supported by QEMU; DMTCP can checkpoint KVM/QEMU virt. machine (Caveat: dynamically interpreting between CPU architectures is 1,000 times slower.)