Transparent Checkpoint-Restart: Re-Thinking the HPC Environment

Gene Cooperman
gene@ccs.neu.edu

College of Computer and Information Science
Northeastern University
Boston, United States

Aug. 20, 2015

*Partially supported by NSF Grant ACI-1440788, by a grant from Intel Corporation, and by an IDEX Chaire d'Attractivité (U. of Toulouse/LAAS).

Checkpointing for HPC at Two Extremes

Part I: Supercomputers

Part II: Many cores on a single computer

DMTCP: Distributed MultiThreaded Checkpointing

DMTCP provides transparent checkpoint-restart (saving/restoring a computation) **without any modification to the application binary, to the run-time libraries, to the operating system**.
*(***Portability across MPIs:** *works independently of MPI implementation.*
Based on standards: *POSIX system calls, Linux proc filesystem.*
Enhanced portability: *no need to modify lower software layers.)*

DMTCP works on **any language** (C/C++, Java, Python, Perl, Matlab, bash shell, MPI, etc. **+ UPC/PGAS (since 2014)**).
(They're all just binary executables! DMTCP works at the level of machine language.)

DMTCP demonstrated to work on PGAS (UPC) at HPDC-14, Cao et al.; and on MIC standalone
(In principle, should work on today's MVAPICH2-X (MVAPICH+PGAS); and on next year's MVAPICH2-MIC (with MIC on the motherboard))

Outline: A Talk in Two Parts (part 1)

1. Plugins as a prerequisite for supporting checkpointing in HPC.
   Plugins must inter-operate with:
   1. Major MPI implementations: (MVAPICH2, Open MPI, Intel MPI, MPICH2)
   2. Resource managers: (e.g., SLURM, Torque, LSF)
   3. MPI process managers: (e.g, Hydra, PMI, `mpispawn`, `ibrun`)
   4. The network: InfiniBand; sockets; newer APIs
   5. Other computation models: OpenSHMEM, PGAS
   6. Each new version of Linux kernel
2. Replacing the batch *queue* of HPC with a batch *pool* for a many-core CPU
   - ...

Outline: A Talk in Two Parts (part 2)

1. Plugins for Supercomputing — transparently inter-operate with:
   1. ...
2. Replacing the batch *queue* of HPC with a batch *pool* for a many-core CPU
   1. Batch queues assume that we execute a process from beginning to end.
   2. Using checkpoint-restart, a new job is executed beyond the initialization phase. It is then checkpointed and the checkpoint image is added to the batch *pool*.
   3. *QUESTION:* In a many-core computer, how does one decide which processes to run together?
   4. *PARTIAL ANSWER:* Do trial runs of different combinations of jobs chosen from the batch pool, and use hardware performance counters to measure which combination has the highest throughput.

Part I: Checkpointing for Supercomputing: The Secret is in the Plugins

(For the latest status, see Friday's talk,
"Transparent Checkpointing for Supercomputing",
Jiajun Cao and Rohan Garg.)

A Short Demo

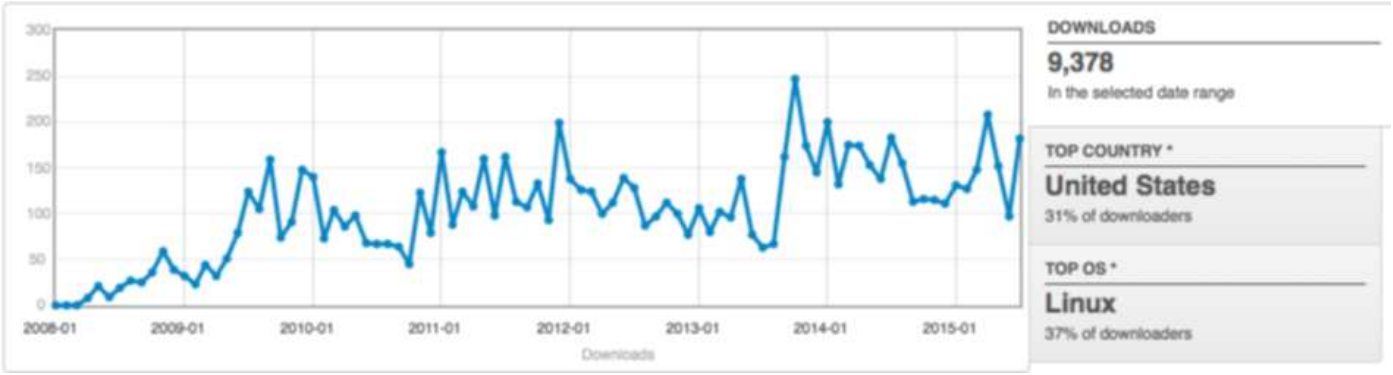
- As easy to use as:

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

- The project is now 11 years old.
- A Quick Demo!
- `http://dmtcp.sourceforge.net/plugins.html`

DMTCP is Mature Software

- Published literature: more than 40 other groups (not us) using DMTCP in their work and published in the years 2011–2015.
- *Start here:* FAQ (42 questions/answers): `google DMTCP FAQ`
- *Downloads:*



- *DMTCP Forum:*

2009	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct (1)	Nov	Dec
2012	Jan (5)	Feb (24)	Mar (22)	Apr (24)	May (4)	Jun (8)	Jul (14)	Aug (20)	Sep (14)	Oct (20)	Nov (5)	Dec (1)
2013	Jan (6)	Feb (1)	Mar (32)	Apr (6)	May (3)	Jun (26)	Jul (14)	Aug (11)	Sep (15)	Oct (44)	Nov (33)	Dec (2)
2014	Jan (9)	Feb (19)	Mar (12)	Apr (5)	May (2)	Jun (4)	Jul (3)	Aug (2)	Sep (8)	Oct (32)	Nov (30)	Dec (8)
2015	Jan	Feb (18)	Mar (31)	Apr (41)	May (33)	Jun (13)	Jul (15)	Aug (6)	Sep	Oct	Nov	Dec

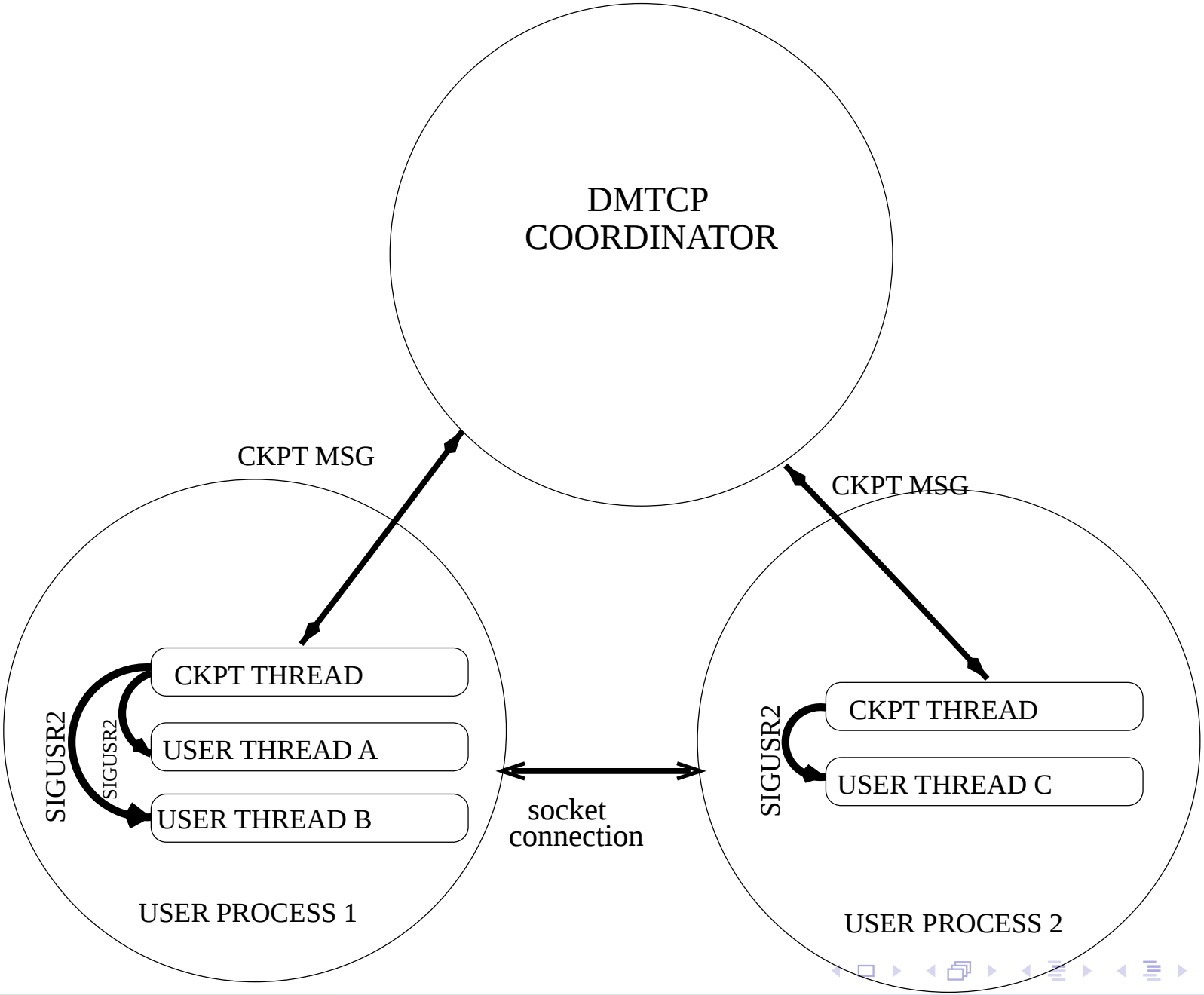
But How Does It Work?

- `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.

DMTCP Architecture



What Happens during Checkpoint?

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.

Plugins

WHY PLUGINS?

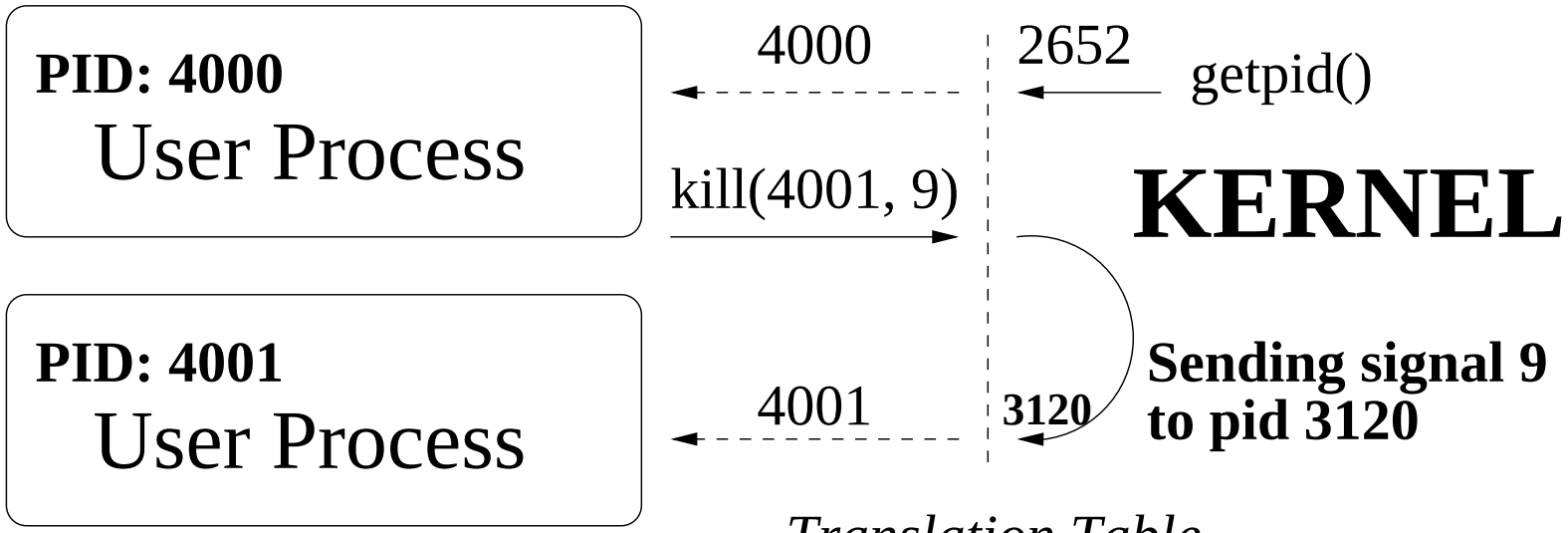
- New computer host: new pathnames, new mount point, new IP address
- The DISPLAY environment variable must be changed on new host.
- DB: Disconnect from database server at ckpt; re-connect on restart.
- Authentication: Note authentication key used by app; re-use on restart.

NOTE: For heterogeneous systems, ckpt-restart is not well-defined.

- "Just restore it the way it was." is not well-defined.
  Simple example: What if we checkpointed in the middle of `sleep()`
  1. Skip the rest of the sleep on restart:
     Maybe we were sleeping, to give the user time to respond.
  2. Continue to sleep on restart:
     Maybe one thread was sleeping, to give a second thread enough time for it to finish.

A Simple 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

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.

Some Plugins Distributed with DMTCP

Top-level directories in the DMTCP distribution. (Lines of code in parentheses, as measured by sloc, including support code, test suites, etc.)

- `ls plugin`
  batch-queue (6,000, incl. test suite), modify-env (237), ptrace (1,000)
- `ls contrib`
  python (202), infiniband (9,000), ib2tcp (1,000), ckptfile (37), ckpttimer (161), apache (73), condor (289), kvm (1,800), tun (596)

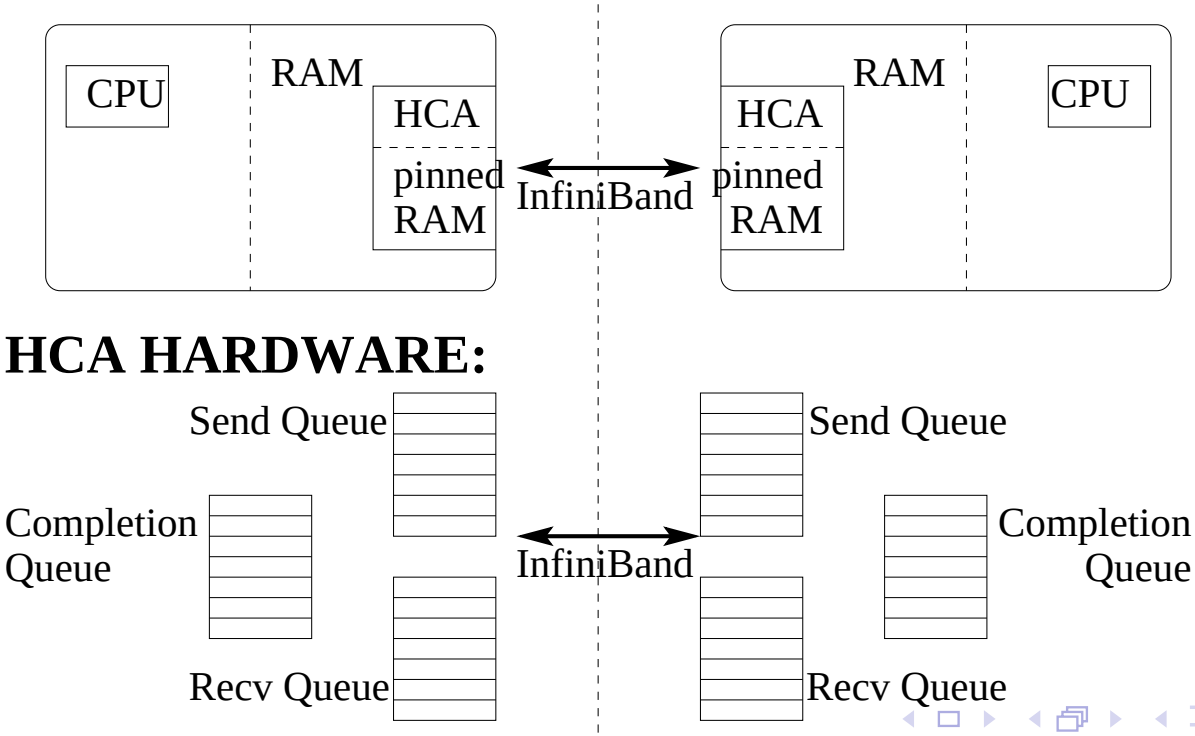
InfiniBand Plugin

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!

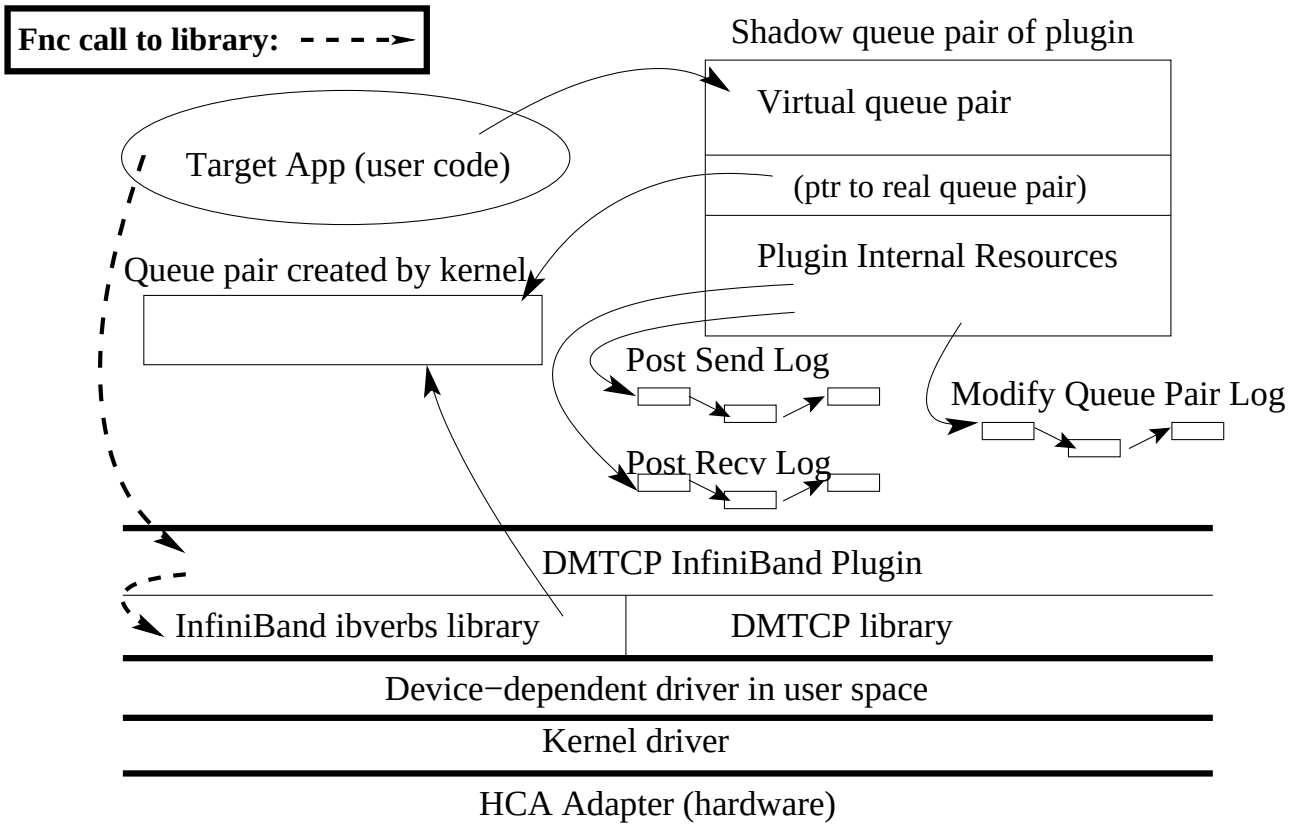
socket plugin: `ipc/socket`; *InfiniBand plugin:* `infiniband`

- 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

Batch-Queue Plugin: Resource Managers

Handles the plumbing to launch and to restart a DMTCP-based batch job.

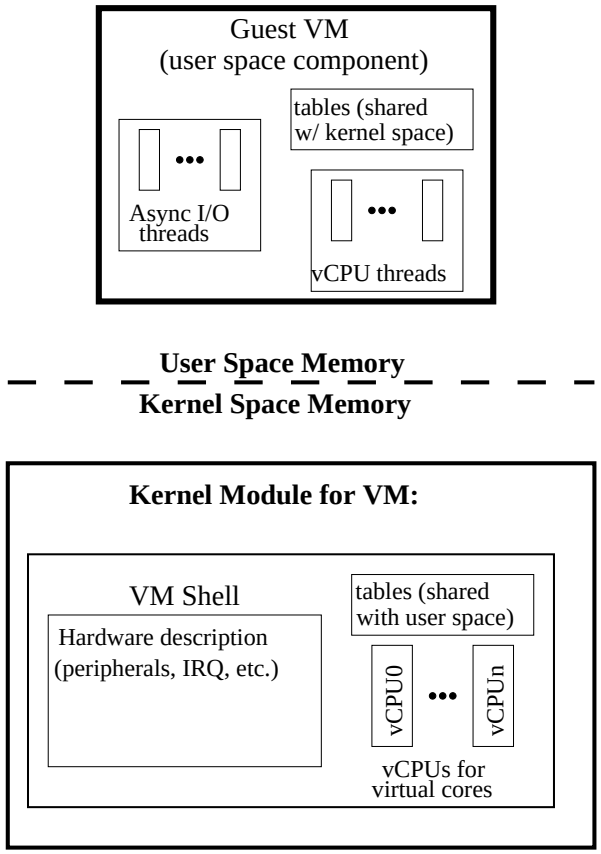
For example, the plugin will temporarily disable the resource manager connection during checkpoint, and re-enable it during restart. (The connection to the resource manager represents an "external connection", since the resource manager process itself is not being checkpointed — only the MPI application process. So, we must disconnect prior to checkpoint.)

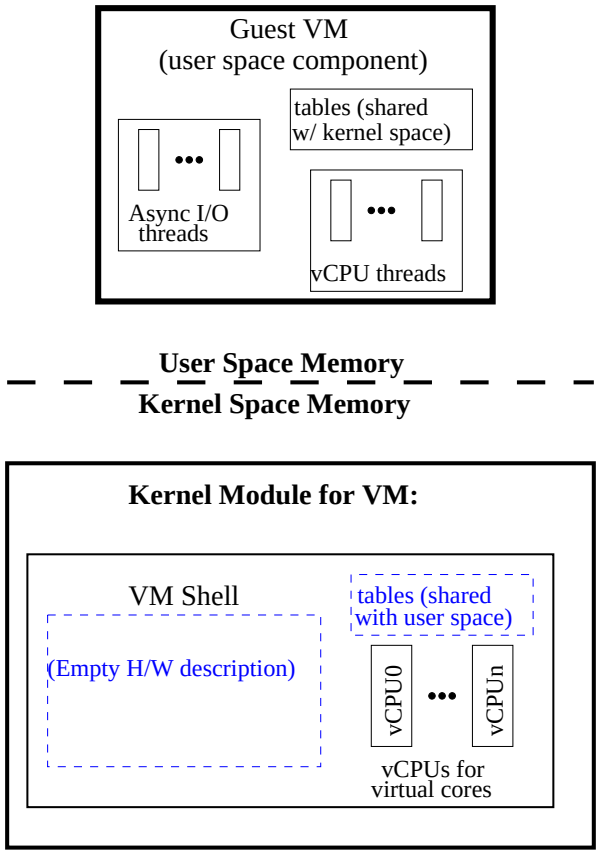
In another example, the resource manager on a computer node will have information on which MPI processes were located on that node. This is important, since two MPI processes on the same node may be using shared memory. It's important, at restart time, to co-locate MPI processes on the same node, if they were co-located prior to checkpoint.

The resource manager remains unaware of DMTCP. No modifications to the resource manager are required.

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`

Some Collaborations with Additional Groups

1. **CLOUD**: *Checkpointing as a Service in Heterogeneous Cloud Environments* (CCGrid'15) (with Matthieu Simonin, Christine Morin, Jiajun Cao);
   Demonstrated to work both on Snooze and OpenStack
2. **BIG DATA** (in progress): *Checkpointing Hadoop jobs:* building on Chronos system of Shadi Ibrahim and his collaborators to enable long-term checkpointing (e.g., suspend current Hadoop job to allow high priority job to execute)
3. **HaaS (Hardware as a Service)** (in progress):
   Novel cloud service: offer rapid access to custom platforms; with Mass. Open Cloud (with Orran Krieger, Peter Desnoyers, Apoorve Mohan)
   Use "kexec" for fast booting to another Linux; followed by ckpt/restart of theinit process

Part II: Batch Pools: A New Type of Batch Queue

Work in progress: Many-Core computers

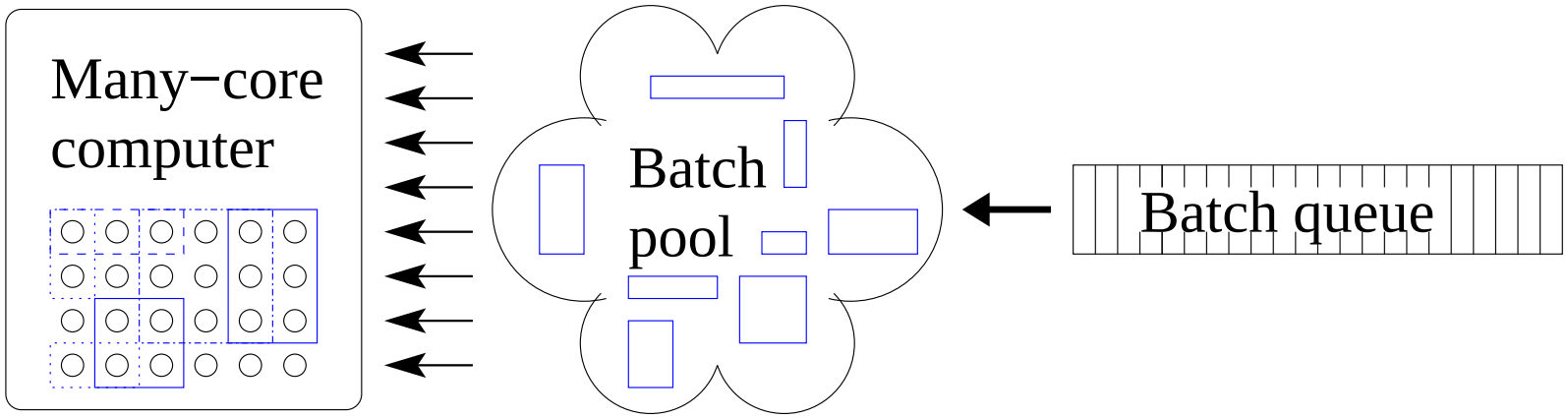
(If successful, extend to a parallel queue with MVAPICH2?)

"Batch Queues" versus "Batch Pools"

- Historically, a resource manager system would allow a batch job to reserve a fixed number of computer nodes. Each computer node was allocated exclusively to that job and no other.
- Currently, a resource manager allows a batch job to exclusively reserve CPU cores (e.g., CPU affinity mask) instead of the entire computer node.
- This requires each job to estimate (or more often over-estimate) the number of CPU cores required.
- Providing greater throughput through dynamic sharing of computer nodes is difficult. But greater throughput through dynamic sharing of the CPU cores of a single node (over-commitment of cores) is easy.

We're entering an era of many-core computers, and we're leaving spare CPU cycles are falling on the floor!

The Architecture of a Batch Pool



Goal: dynamic over-commitment of CPU cores by threads of multiple jobs

Secondary Goal: matching compatible jobs

(For example, mixing a CPU-bound job with a RAM-bound job on the same core.)

Issue: The throughput is measured by instructions per second divided by CPU cycles per second.

We need an aging policy, or else some jobs might never run.

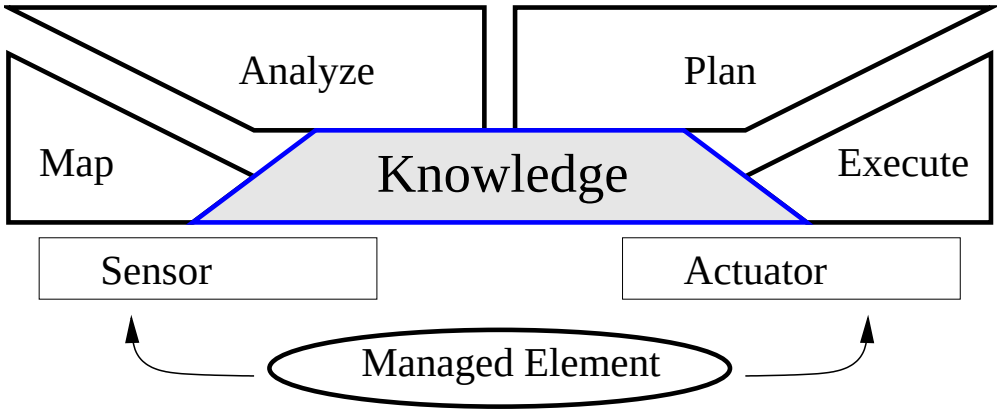
Proposal: Batch Queue/Pool for a Single Many-Core CPU

1. While the batch pool is below some threshold, draw the next job from the batch queue, execute for a fixed period of time (to get past the initialization stage to a steady-state regime), and checkpoint. Save the checkpoint image in the batch pool.
2. Periodically re-balance which jobs in the batch pool will execute:
   1. Checkpoint all currently running jobs, and save the checkpoint images into the batch pool.
   2. For each checkpoint image currently in pool, run it for a little while, to compute job characteristics in steady state.
   3. Select a fixed number of candidates for combinations of batch jobs to run in parallel *(see next slide)*. (The job characteristics above are inputs for selecting good candidates for batch jobs to run in parallel.)
   4. Test each candidate to measure throughput (instructions per second divided by CPU cycles per second), as biased by aging.
   5. Select winning candidate; execute until the next time interval.

Autonomic computing: MAPE-K

Autonomic Computing: Analogy with autonomic nervous system: The brain provides high-level control. Low-level processes are controlled autonomously, using "knowledge" from the brain.

MAPE-K: Monitor, Analyze, Plan, Execute; and Knowledge



(collaboration with Saïd Tazi, LAAS-CNRS and U. of Toulouse, France)

Scenario: an Autonomic Batch Pool

1. **Initial heuristics** (to be determined)
2. Aging: Raise or lower priority based on if the job ran in the last epoch
3. The autonomic computing mechanism does the low-level tuning: **throughput** (instructions executed for all jobs on a node, divided by CPU cycles for all cores); **limiting cores** (core affinity); **hyper-threading** (selectively turning it on for individual jobs); **aging** (guaranteeing progress for each job)
4. System administrators set the high-level goals: high throughput, low energy use, absolute and relative job priorities, soft or hard deadlines, fairness policies, ...

Questions?

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

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

QUESTIONS?

DMTCP Resources

- DMTCP FAQ:
  `http://dmtcp.sourceforge.net/FAQ.html`
- Architecture of DMTCP:
  `http://sourceforge.net/p/dmtcp/code/HEAD/tree/trunk/doc/ar`
  (`.../trunk/doc/architecture-of-dmtcp.pdf`)
- Plugin Tutorial:
  `http://sourceforge.net/p/dmtcp/code/HEAD/tree/trunk/doc/pl`
  (`.../trunk/doc/plugin-tutorial.pdf`)
- Plugin Examples:
  `http://sourceforge.net/p/dmtcp/code/HEAD/tree/trunk/test/p`
  (`.../trunk/test/plugin/`)

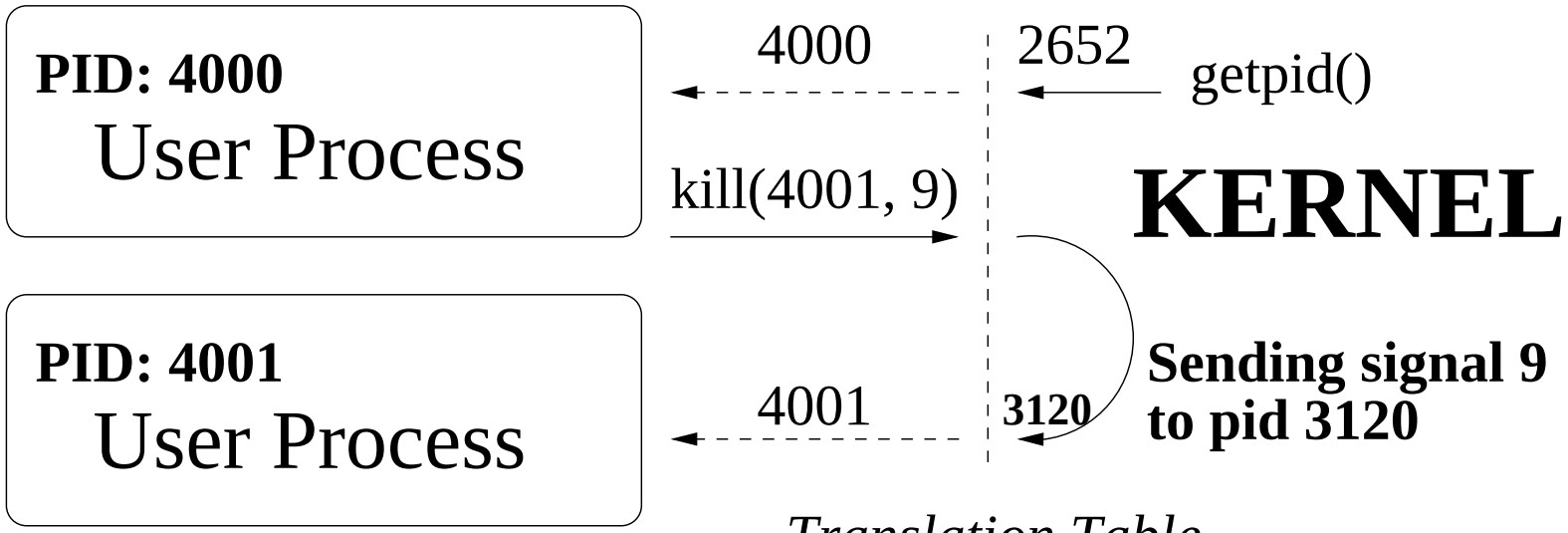
Some use cases for checkpoint-restart: fault tolerance; fast startup (ckpt after initialization); process migration; save/restore of workspace (for interactive sessions); debugging (last ckpt before bug); the ultimate bug report; ...

Porting DMTCP to Android

1. Porting DMTCP checkpointing software from Linux to Android — transparently inter-operate with:
   1. Bionic libc (The Android standard libc is not GNU libc.)
   2. Binder (a different model of launching processes)
   3. Android kernel extensions (Ashmem kernel driver, for shared memory; used by Binder)
   4. Service Manager (process asks for services of other processes through service manager)
   5. Dalvik virtual machine (similar to Java JVM); now replaced by ART (Android RunTime)

A First 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

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