DMTCP: System-Level Checkpoint-Restart in User Space

Kapil Arya[1], Gene Cooperman[1](presenting)

{kapil,gene}@ccs.neu.edu

College of Computer and Information Science
Northeastern University

August 26, 2014

[1]This work was partially supported by the National Science Foundation under Grants ACI-1440788, OCI 1229059 and OCI-0960978, and by a grant from Intel Corporation.

DMTCP: Distributed MultiThreaded CheckPointing

- Transparent Checkpoint-Restart
  - No modifications to the application program
  - Works on any language (C/C++, Java, Python, Perl, Matlab, vim, bash shell, MPI, VNC server, etc.). *(They're all just binary executables!)*
  - Checkpoints initiated externally, or by the application.
- **Works in User Space**
  - No modifications to the kernel (no kernel module)
  - Stay close to standards: most O/S access through POSIX syscalls
  - Supports Intel (x86, x86_64) and ARM (armv7, armv8: 64 bits)
- Plugin architecture
  - Allows for third-party plugins, modular development
- The project is now 10 years old
  - Most widely used transparent checkpointing package in user space??

Using DMTCP

- As easy to use as:

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

- For MPI applications:

  `dmtcp_launch mpirun_rsh` *[mpi_flags]* `./mpihello`

  (but plugins also make it easy for top-level MPI to call DMTCP:
  Example: *see DMTCP plugin, batch-queue, for SLURM and Torque)*

- Freely available: `http://dmtcp.sourceforge.net`
  - $\approx$ 2,000 downloads per year as source tarballs
  - Available in major Linux distros: unknown number of "downloads"
  - Active user community (incl. academia, industry):
    `http://sourceforge.net/p/dmtcp/mailman/dmtcp-forum/`

DMTCP Internals

- `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.

IMPLEMENTATION: About 27,000 lines of code (including about 100 lines of assembly).

Three Generations of DMTCP

Generation 1: Single process (multi-threaded)

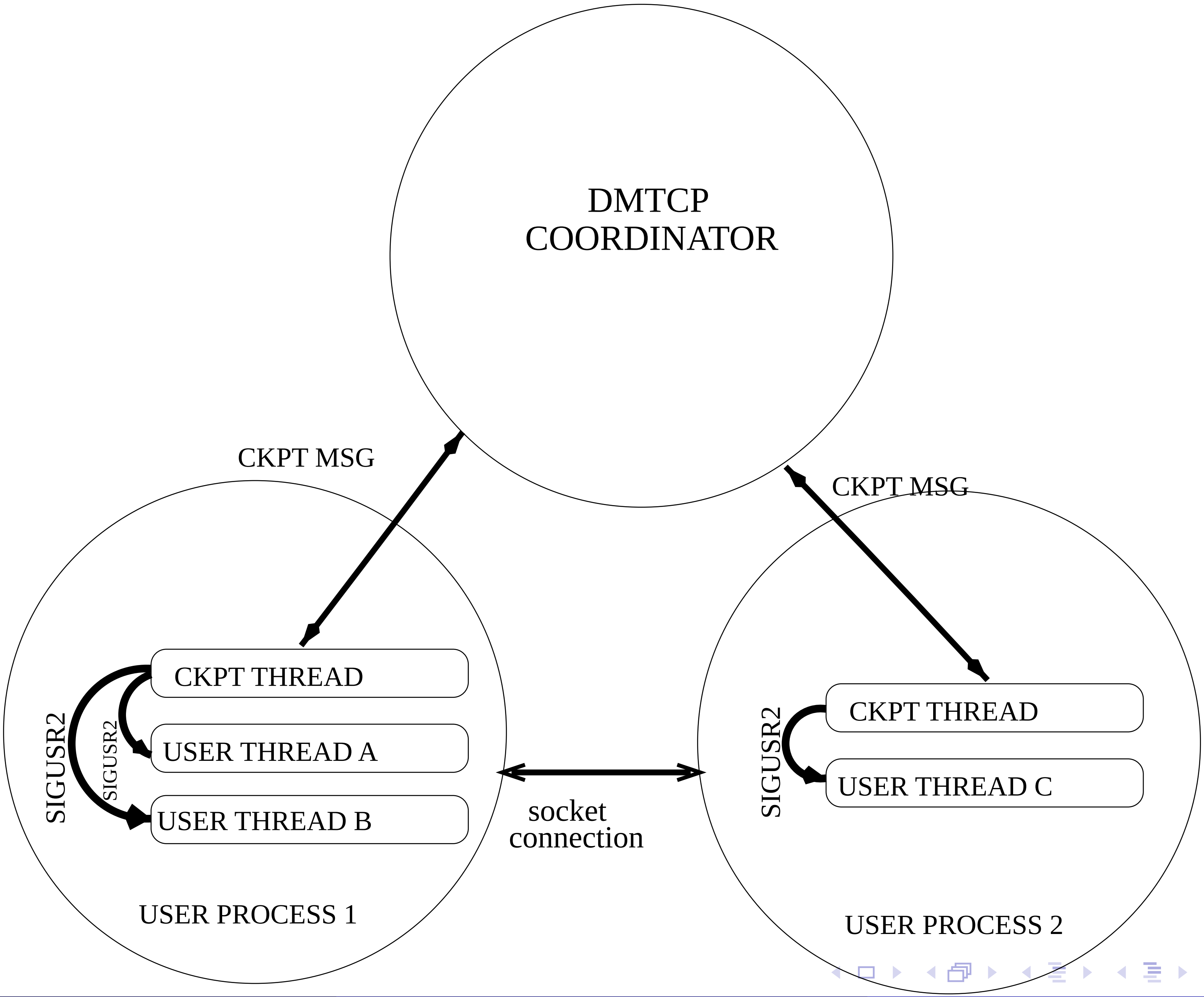
Generation 2: Distributed processes (support for most POSIX calls, and for TCP/IP: handle common case of Ethernet hardware)

Challenges: InfiniBand hardware, Programmable GPU hardware (for OpenGL), Intel Xeon Phi hardware, Resource Managers (e.g., SLURM, Torque) : run-time management of hardware); Virtual Machines (e.g., KVM/QEMU: simulation of hardware)

Generation 3: DMTCP-2.x (plugins to adapt to external resources)

Solutions: Plugins now exist for each of the above. (Exception: Xeon Phi supported through an alternative DMTCP build)

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 from the earlier slide?)
5. Now the checkpoint thread can copy all of user memory to a checkpoint image file, while the user threads are blocked.

But What about: the O/S Kernel State, the Network, ...?

Recall: We copy all of user-space memory to a ckpt image file. Any extra information can also be stored in user-space memory.

- *Problem:* Some of the process state is in the kernel, such as open file descriptors and Network sockets.
- *Solution:* Use POSIX system calls to interrogate and save the state during checkpoint and later restore the state during restart.

- *Problem:* Some of the state is in network hardware (data in flight).
- *Solution:* "Drain the network" at the time of checkpoint.
  1. At the send endpoint of each socket, send a unique "cookie".
  2. At the receive endpoint of each socket, receive all data until the "cookie" is seen.
  3. At resume-time or restart-time, pass network data back to the send endpoint, and push back into network.

Two Keys to Supporting Checkpointing for MPI

1. **Plugins**

   Challenge (modular code): Avoid `#ifdef` for each special case (InfiniBand, ptrace (GDB), record-replay, ...).

   Challenge (multiple developers): **Allow each developer to make arbitrary modifications to DMTCP!**

   Each developer adds a callback to their own plugin during different events: application startup, pre-checkpoint, write checkpoint image, resume, restart, new child process, new thread, child exit, thread exit.

   Challenge (ssh): First correct implementation for *ssh* connections.

2. **InfiniBand plugin**

   Shadow device driver: Applications see shadow structs; plugin passes info between shadow and actual InfiniBand struct.

   Drain the network: Extend TCP/IP-based DMTCP technique for distrib. checkpoint-restart. **(IB network stays alive!)**

   Very complex protocol: Isolate InfiniBand complexities from general checkpoint-restart complexity. *(If one can do it for InfiniBand, any other protocol will seem easy!)*

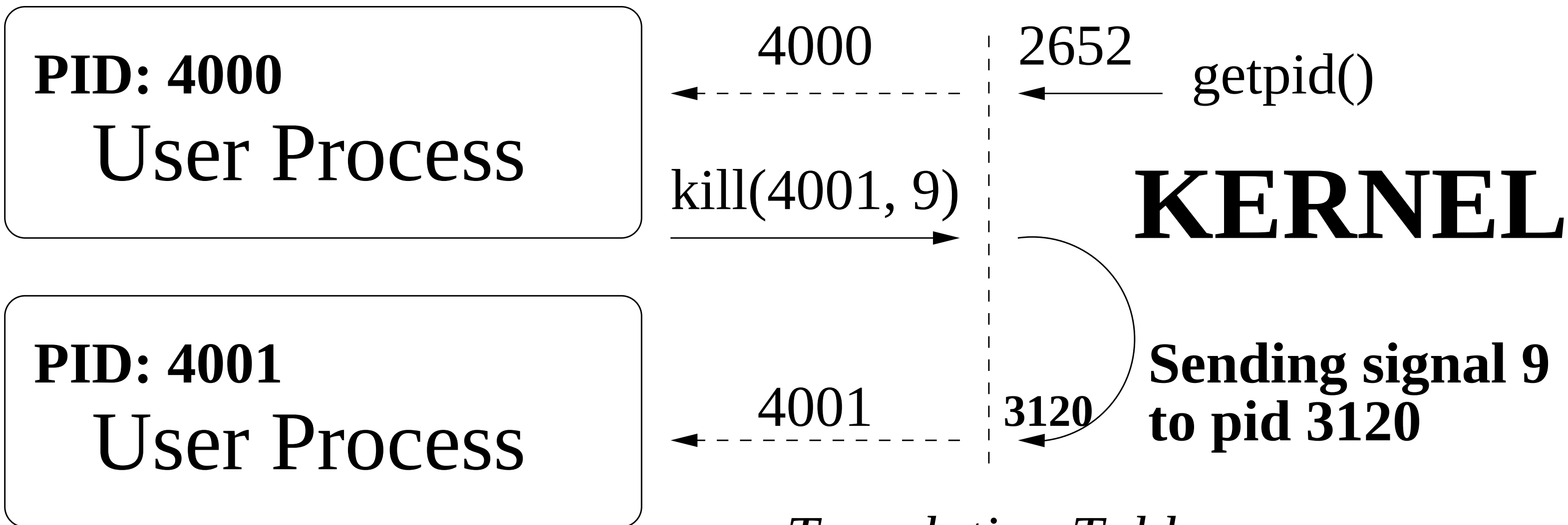
KEY #1 (supporting MPI): Plugins

WHY PLUGINS?

- New computer host: new pathnames, new mount point, new IP address
- DB: Disconnect from database server at ckpt; re-connect on restart.
- Authentication: Note authentication key used by app; re-use on restart.
- Re-configure application (e.g., different DISPLAY environment variable on restart)

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

IMPLEMENTATION: Wrapper function around each syscall using pid (47 functions, about 10 lines each).

More Complex Plugins

Extending checkpoint-restart to complex, new domains is nearly impossible in practice, without the use of plugins.

- A few success stories using plugins:
  1. Transparent checkpointing of InfiniBand
     "Transparent Checkpoint-Restart over InfiniBand",
     Jiajun Cao, Gregory Kerr, Kapil Arya, Gene Cooperman, HPDC-14
  2. Checkpointing a networked group of virtual machines
     "Checkpoint-Restart for a Network of Virtual Machines",
     Rohan Garg, Komal Sodha, Zhengping Jin and and Gene Cooperman,
     IEEE Cluster–2013
  3. Transparent checkpointing of 3D-graphics
     "Transparent Checkpoint-Restart for Hardware-Accelerated 3D Graphics"
     Samaneh Kazemi Nafchi, Rohan Garg, and Gene Cooperman
     `http://arxiv.org/abs/1312.6650` (work still in progress)
  4. Checkpointing of GDB sessions

Typical Sizes of Plugin Codes

Plugin	Language	Lines of Code	Wrapper fncs
Internal Plugins			
Socket	C/C++	1,356	17
File	C/C++	2,276	48
Event	C/C++	909	12
SysVIPC	C/C++	1,154	14
Timer	C/C++	419	14
SSH	C/C++	1,021	3
Pid	C/C++	1,644	47
Contributed Plugins			
Ptrace (GDB)	C/C++	938	7
KVM	C	749	2
Tun	C	351	3
RM (Slurm/Torque)	C/C++	1,715	13
InfiniBand	C	2,700	34
OpenGL	C/C++	4,500	119
Application-Specific Plugins			
Malloc	C/C++	116	10
Dlopen	C/C++	28	2
Modify-env	C	134	0
CkptFile	C/C++	37	0
Uniq-Ckpt	C/C++	39	0

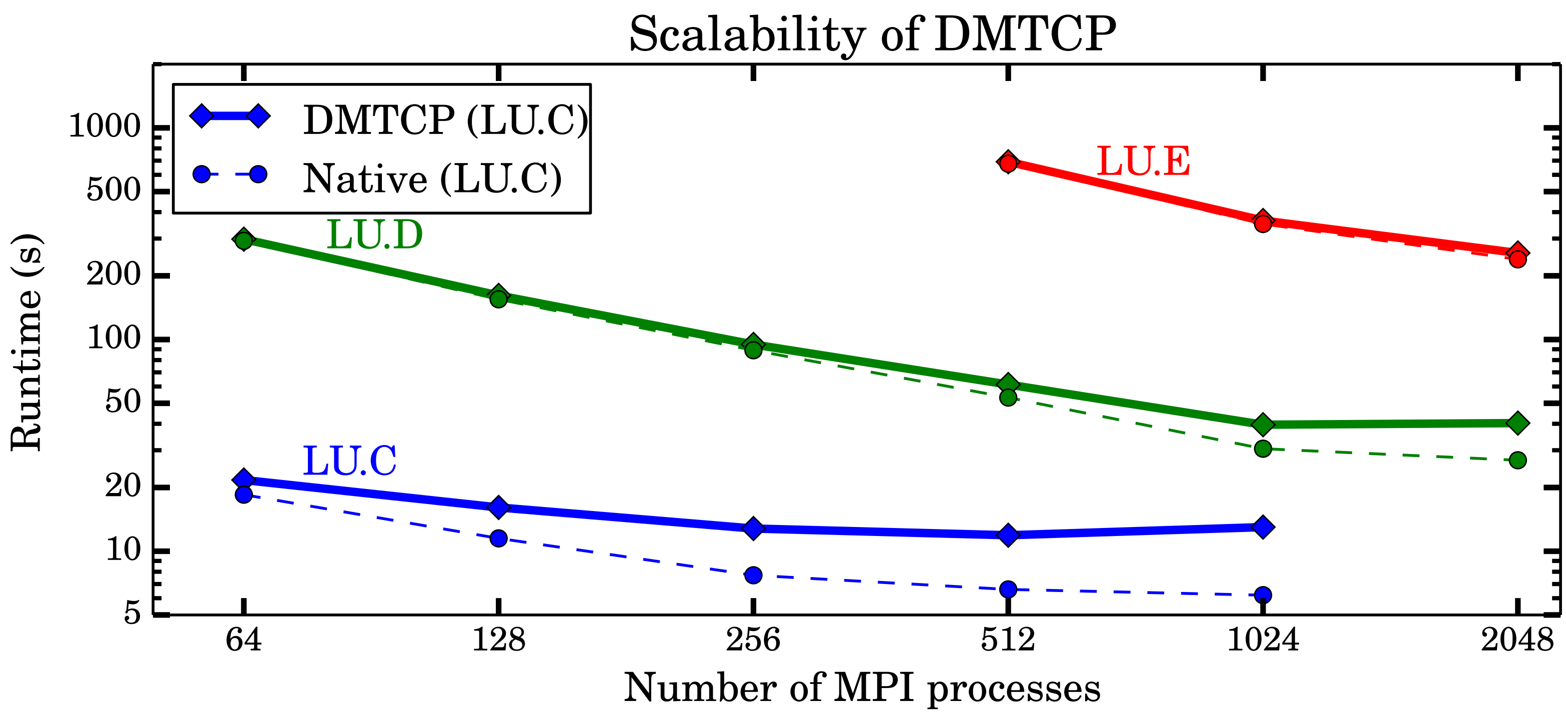
KEY #2 (supporting MPI): Checkpointing InfiniBand

- MPI-based (MVAPICH/BLCR, etc.): MPI calls single-node checkpointing package (e.g., BLCR)
  - Not available for non-MPI computations, such as pure PGAS
- Traditional MPI-based Checkpointing of an InfiniBand computation involves:
  - "Tear down" InfiniBand on checkpoint; re-build on resume
  - Checkpoint individual processes
  - Recreate InfiniBand network
- **Solution proposed w/ DMTCP (direct distributed checkpointing)**
  - DMTCP runs on top: MPI is just another distributed application `dmtcp_launch mpirun_rsh ./mpihello arg1 ...`
  - **Don't tear down the network!** (Save time during the common checkpoint-resume sequence.)
  - Build it as a plugin! (more modular)
    - Easy to experiment with alternative algorithms for InfiniBand.
    - Easy to tune an existing InfiniBand plugin.

Checkpointing InfiniBand (strong scalability curves)

Strong scalability: Fix workload and increase number of MPI processes. *Note:* DMTCP has very small overhead, except for runs below 50 s (see y-axis).

(See next slide for analysis of startup time vs. runtime overhead.)



Checkpointing InfiniBand (derived startup, runtime overhead)

Derived startup and runtime overhead times, based on previous NAS LU benchmark timings:

# processes (running LU)	NAS classes	s: Startup overhead (sec)	r: Runtime overhead in %
64	C, D	3.1	0.8
128	C, D	4.4	1.5
256	C, D	5.0	0.9
512	D, E	7.6	1.0
1024	D, E	8.7	1.3
2048	D, E	12.9	1.7

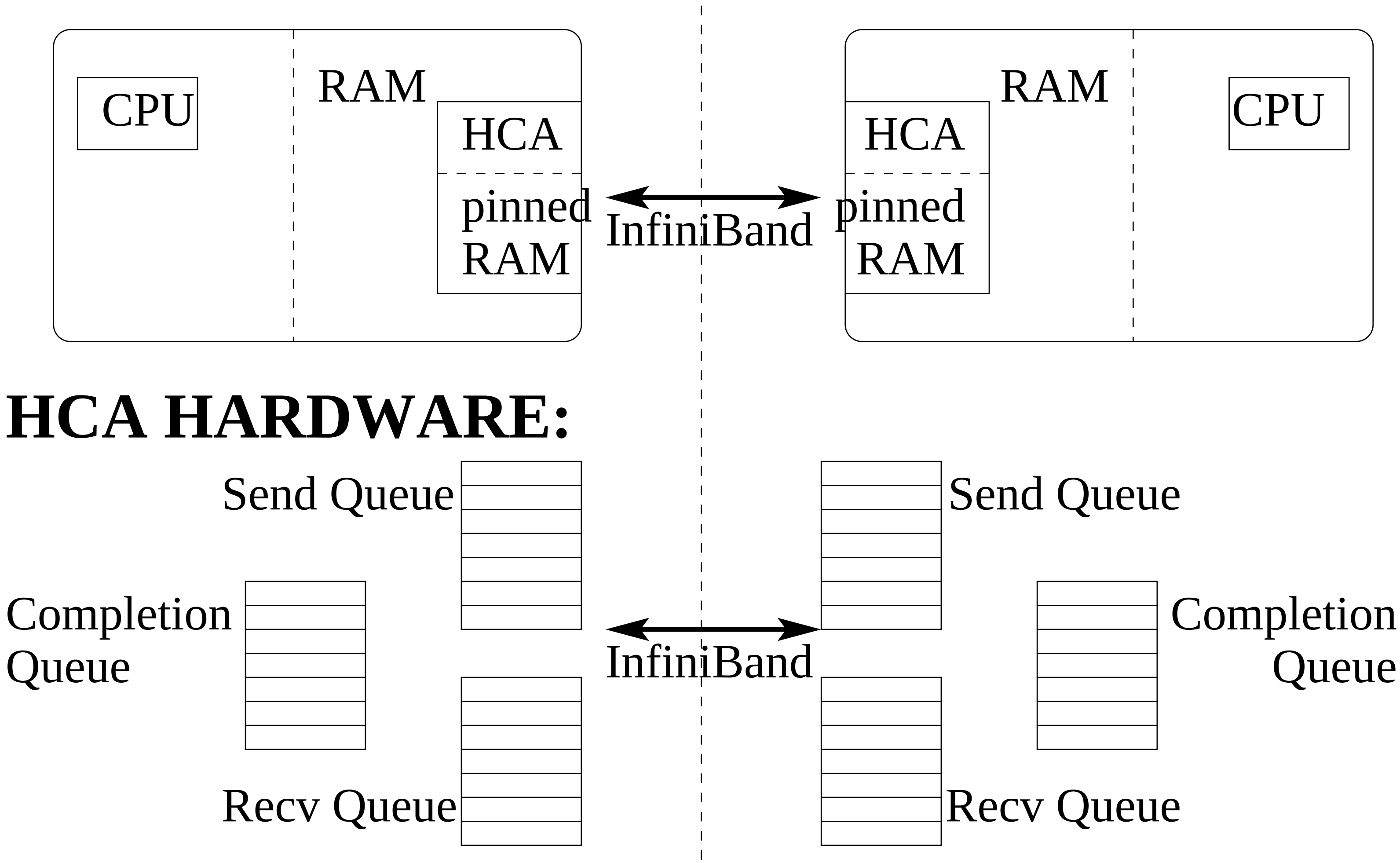
Methodology: Given the native runtimes for two classes of the LU benchmark (e.g., t_1 for LU.C and t_2 for LU.D), and total overhead w/ DMTCP (o_1 and o_2), this yields the implicit startup overhead s and the runtime overhead r:

$$o_1 = s + rn_1$$

$$o_2 = s + rn_2$$

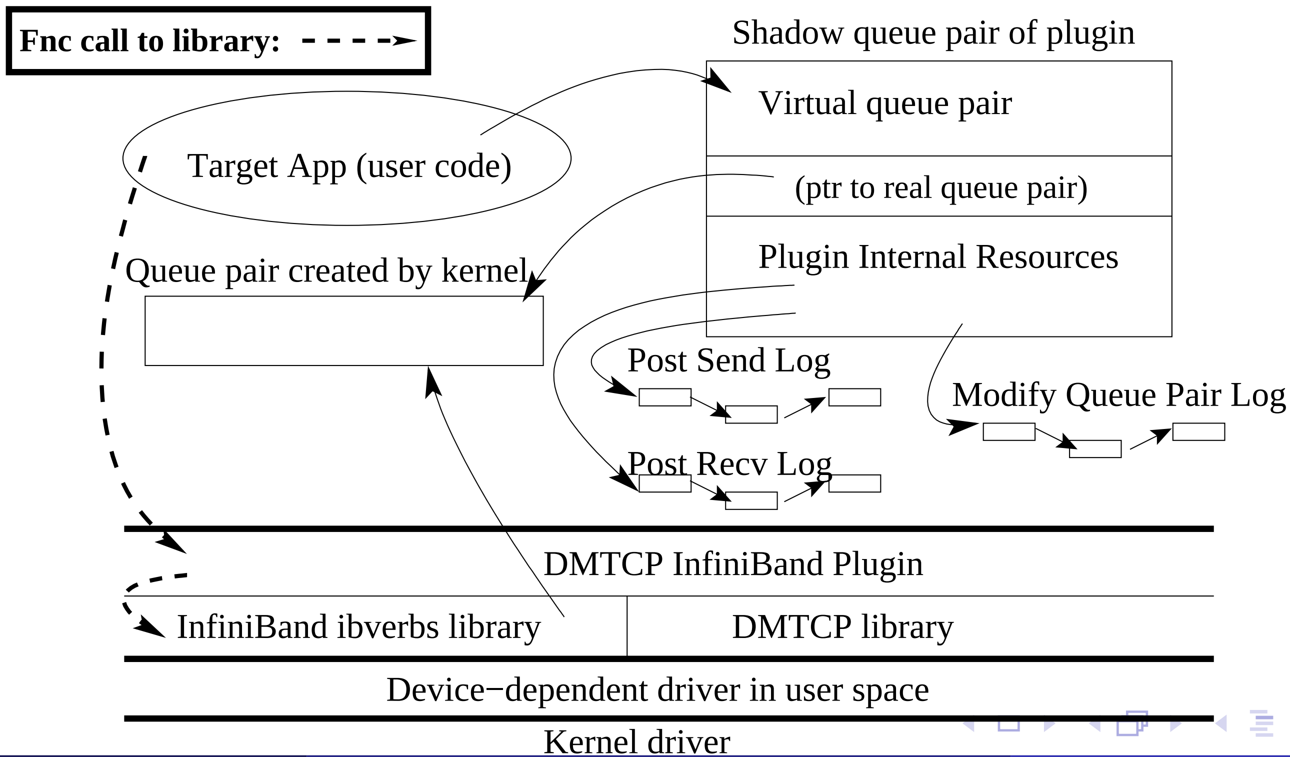
InfiniBand Network (review of concepts)

- InfiniBand uses RDMA (Remote Direct Memory Access).
- RDMA uses *send queue*, *receive queue*, and *completion queue*



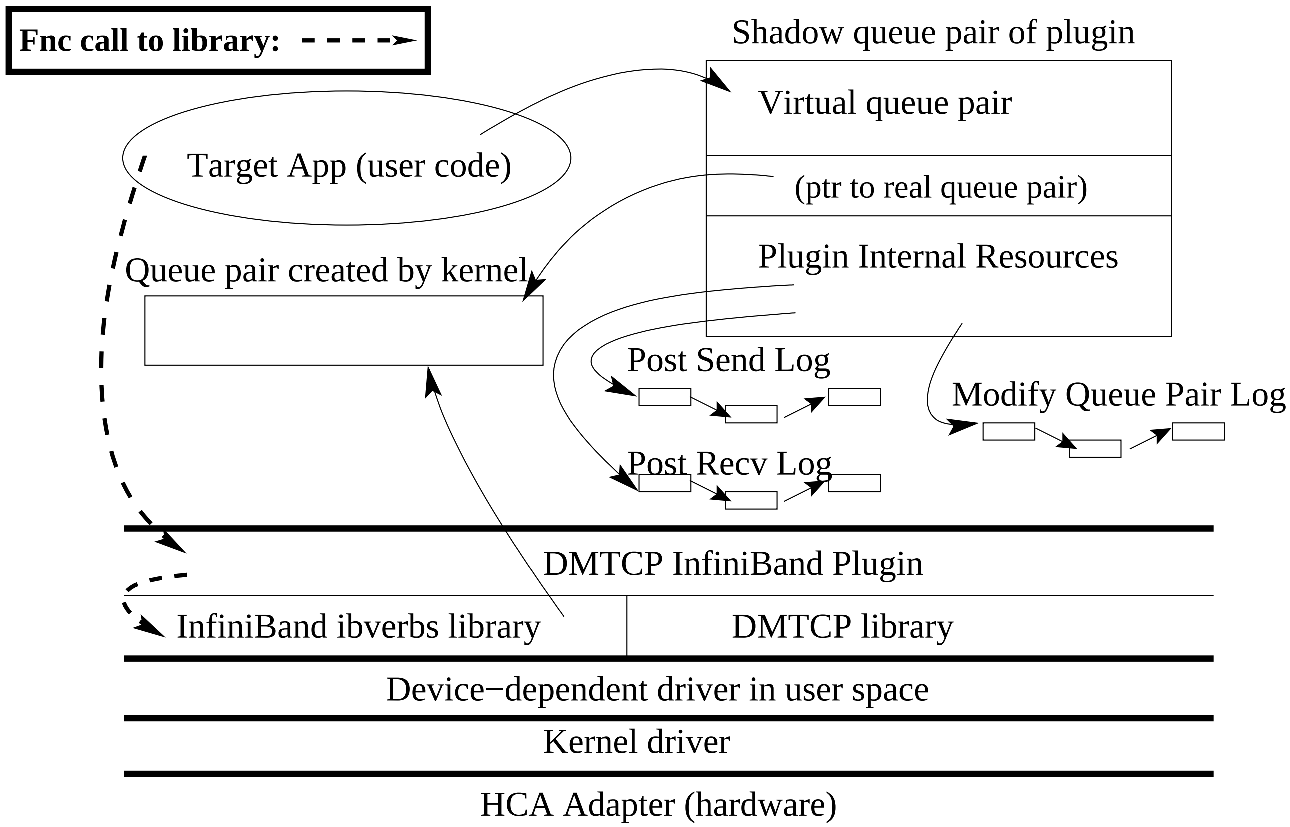
DMTCP and InfiniBand (plugin strategy)

- *ISSUES:* At restart time, totally different ids and queue pair ids. *Some implementations even add "hidden fields" not visible in the struct of the public include file!*
- *Solution:* Give the application a shadow struct, and copy application actions from shadow struct to true InfiniBand struct. (On restart, let InfiniBand create a new "true struct", and re-direct the shadow struct to shadow the new InfiniBand struct.)



DMTCP and InfiniBand (plugin implementation)

- *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.
  - *Total lines of code (infiniband plugin):* 2,700 lines



Transparent Checkpointing of InfiniBand Network (recipe)

Checkpoint:

- Suspend the distributed computation (quiesce user threads)
- Capture the state of the InfiniBand network connections
- Checkpoint each process individually
- Resume the distributed computation

Restart:

- Recreate and restore state of each process individually
- Recreate the InfiniBand network connections
- Restore the state of the InfiniBand network connection
- Resume the distributed computation

Transparent Checkpointing of InfiniBand Network (recipe)

Checkpoint:

- Suspend the distributed computation (quiesce user threads)
- Capture the state of the InfiniBand network connections
- Checkpoint each process individually
- Resume the distributed computation

Restart:

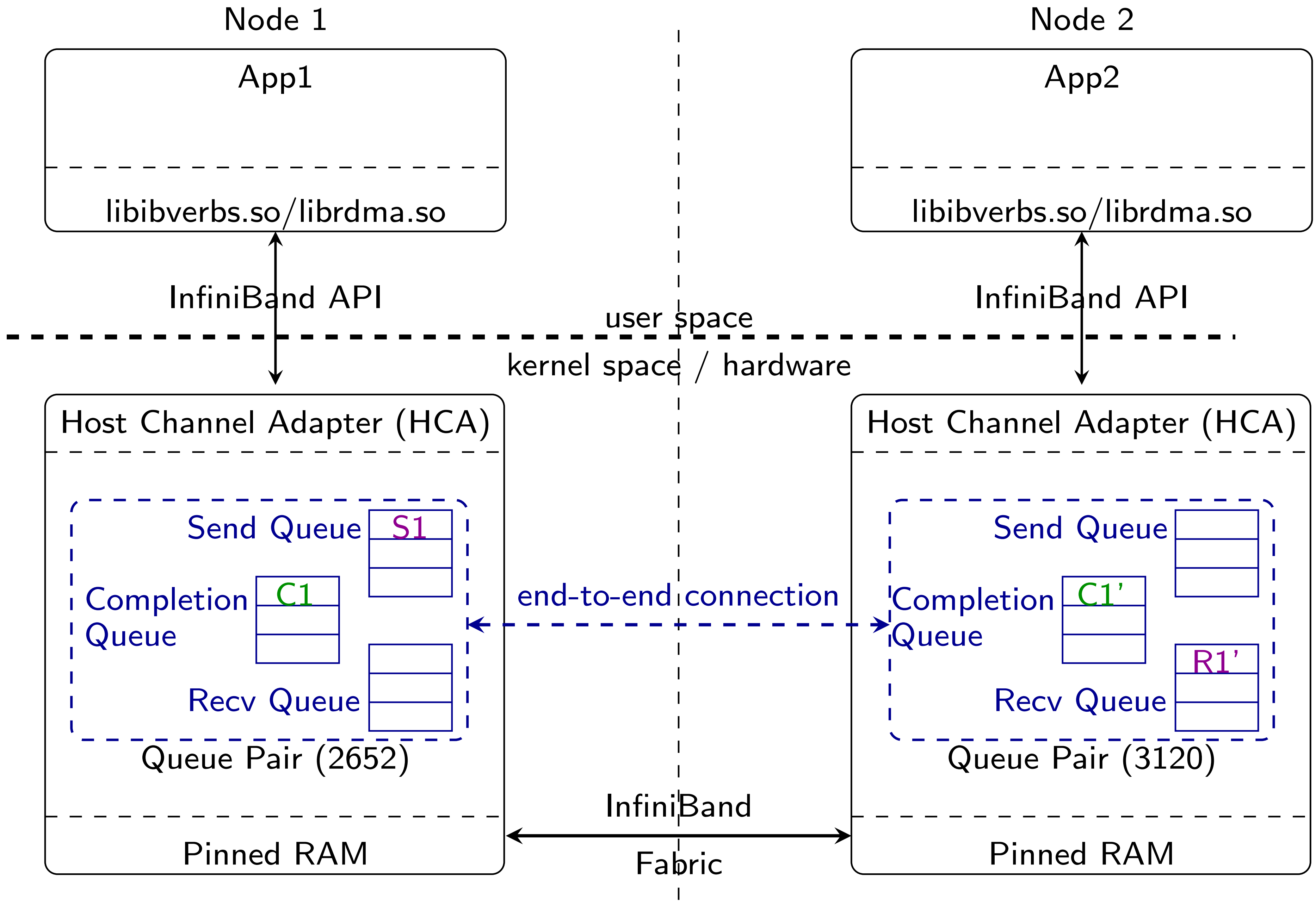
- Recreate and restore state of each process individually
- Recreate the InfiniBand network connections
- Restore the state of the InfiniBand network connection
- Resume the distributed computation

Capture State of InfiniBand Network

Issue: Some of the state is in (proprietary) hardware

- Unfinished send/receive requests
- Un-fetched completion events

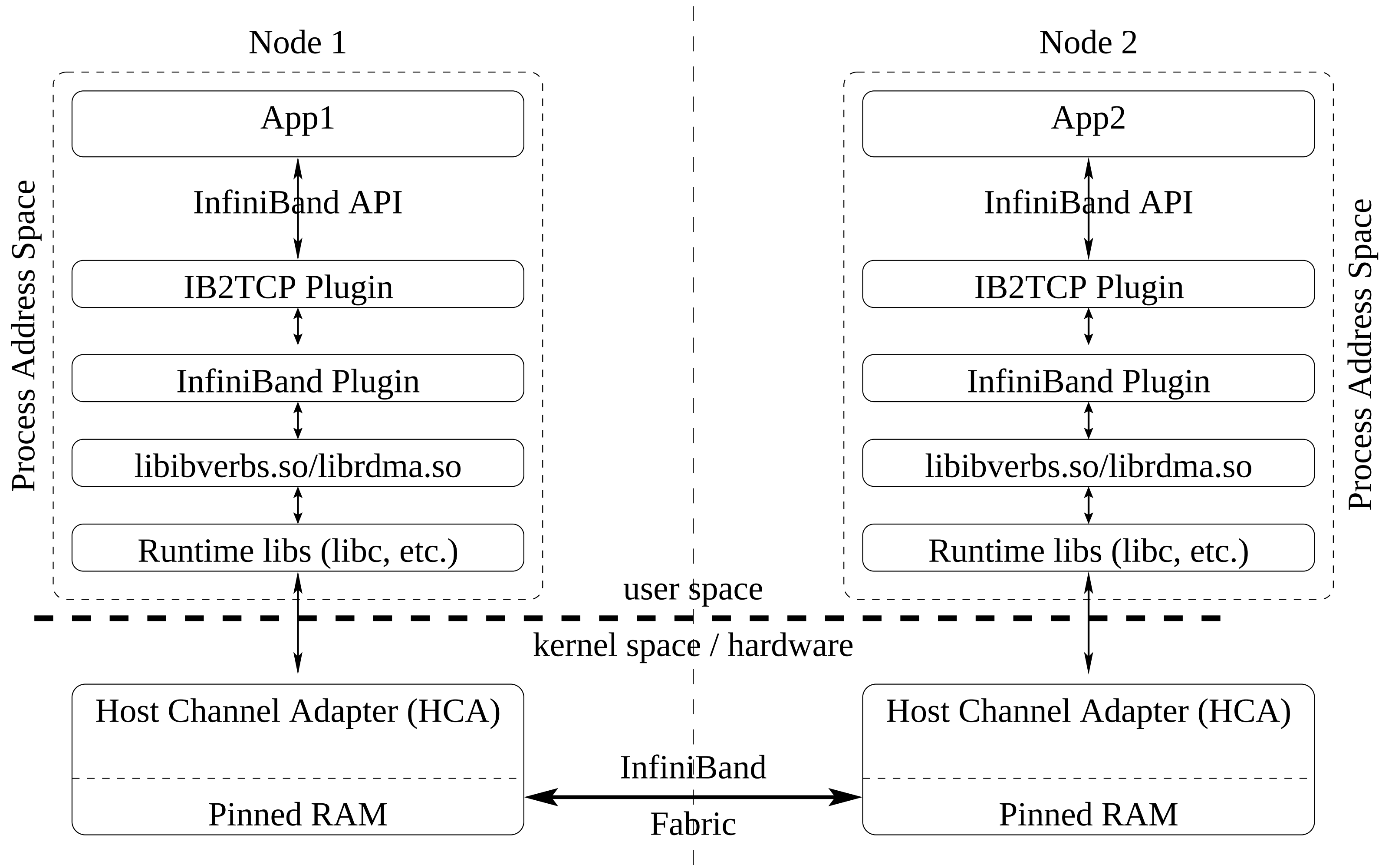
Capture State of InfiniBand Network



Issue: Unfinished Send and Receive Requests

Solution: Virtualize send and receive queues

- InfiniBand plugin intercepts library calls to inspect/modify underlying behavior



Issue: Unfinished Send and Receive Requests

Solution: Virtualize send and receive queues

- Create shadow send and receive queues in process memory
- Intercept post_send() and post_receive() requests to append to shadow queues
- Intercept poll_cq() to remove processed requests from shadow queues
- On checkpoint, record the unprocessed send and receive requests
- On restart, repost un-processed send and receive requests

Issue: Un-fetched Completion Notifications

Solution: Virtualize completion queue

- Drain notification from the completion queue on checkpoint
- Drained notifications are saved in process memory
- On restart, intercept poll requests (poll_cq()) from user code
- Return drained notifications before returning from the hardware

Restore InfiniBand Network State

- Recreate InfiniBand network
- Repost un-processed send/receive requests
- Queue pair ids may change
  - Application remembers the original ids
  - Similarly, memory regions ids may change

Issue: New InfiniBand Ids on Restart

Solution: Virtualize Ids (similar to PIDs)

- Intercept interesting library calls using wrappers
- Assign **virtual** ids for each hardware generated **real** id
- Translation between virtual and real ids
- Update translation table on restart with new ids.

Migrating from InfiniBand to TCP

Bug occurs in production run. Migrate a checkpoint image (prior to the bug) to a local (cheap) Ethernet-based cluster for interactive debugging. Virtualize the InfiniBand hardware:

- Exchange TCP network addresses with InfiniBand peers
- Create TCP sockets between InfiniBand peers
- Create **tcp-send** and **tcp-receive** queues in each process
- Intercept InfiniBand send and receive request
- Append InfiniBand send/receive requests to the **tcp-send**/**receive** queues
- A “send thread” polls the **tcp-send** queue; transmits data over TCP
- A “receive thread” polls the TCP sockets as per **tcp-receive** queue

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.

Thoughts for DMTCP and MVAPICH

- DMTCP (with InfiniBand and batch-queue plugins) currently in beta testing. Hopefully ready for prime time by mid-Fall, 2014.
- Scalable coordinator needed for 100,000 nodes (although, DMTCP's current single coordinator saw minimal overhead in tests with 2,048 MPI processes: 128 nodes $\times$ 16 cores/node)
  PROPOSAL: DMTCP Coordinator plugin
- Integration with resource managers: current support for SLURM, Torque, with LSF planned; plugins can also support other models of integration (e.g., integration with FTB: Fault-Tolerant Backplane)
- Memory cutouts (declare areas of memory not needing to be saved):
  TODAY'S HACK: At checkpoint time, a plugin can zero out a region of memory, and it will be replaced by zero-mapped pages. Principled extension of DMTCP planned for future.

Thoughts for DMTCP and MVAPICH (cont.)

- Heterogeneous restart: DMTCP plugins currently adapt to different network addresses, different pathnames on restart; DMTCP could also share this responsibility with MVAPICH and the resource manager.
- Heterogeneous restart for InfiniBand: restart on different network card (Mellanox vs. Qlogic), or multiple HCA adapters; *(not currently handled, but the current DMTCP design could adapt to these cases)*
- Re-configure on restart (e.g., change DISPLAY on restart for X-Windows: handled by modify-env plugin)
  NOTE: A similar approach could be used to ask MVAPICH to re-configure on restart, based on changed environment variables, or on a callback to MVAPICH.

Optimizations for DMTCP Checkpoint-Restart

Several accelerator options are available with DMTCP for faster checkpoint and restart.

Forked checkpointing: At time of checkpoint, fork a child process. The child checkpoints while the parent resumes computation in parallel.
ISSUE: Many HPC codes use most of RAM. A forked child must rapidly release its memory as it checkpoints, or it will create contention with the parent process.

No dynamic compression: DMTCP calls gzip to dynamically compress memory, as it writes to a checkpoint image.

Fast memory-mapped restart: Use mmap to directly map the checkpoint image into RAM.
Results in demand paging of checkpoint image into RAM.

Differential checkpoint-restart: Incremental checkpoint-restart, and related technologies.

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,
Artem Y. Polyakov, Michael Rieker, Praveen S. Solanki, Ana-Maria Visan

QUESTIONS?