CRD

Berkeley Lab's Checkpoint Restart (BLCR)

Eric Roman and Paul Hargrove

June 17, 2011

http://ftg.lbl.gov/checkpoint

Introduction

Checkpoint.	Save a process's state to a file.
Restart.	Reconstruct the process from a file.
BLCR.	Berkeley Lab's Checkpoint Restart for Linux.
Project goals.	What is BLCR's approach to CR? Why use checkpoint/restart?
System design.	How does BLCR work?
Status.	What does BLCR do now?
Plans.	Where is BLCR going?

Project Goals

Provide checkpoint/restart for Linux clusters running scientific workloads.

Checkpoint and restart shell scripts running MPI applications.

Fit easily into production systems

Run unmodified application source.

Run unmodified binaries. If possible, users should not have to relink codes.

Run on unpatched kernels.

Run with unmodified system libraries. (e.g. libc)

Unrelated features (ptrace, Unix domain sockets) have low implementation priority

Why checkpoint?

We see three main scenarios: scheduling, fault tolerance and debugging.

Scheduling

Preempt running jobs.

Drain queues quickly for maintenance.

Increase system throughput by switching between long and wide jobs.

Increase system utilization by allowing the scheduler to correct earlier decisions.

Gang scheduling. Divide system time up into slots.

Priority scheduling. Run jobs with the highest priority.

Migrate jobs.

Pack jobs for optimal network performance.

Move jobs to faster nodes.

More scheduling flexibility if preemption is used.

Fault Tolerance

Rollback recovery

Not every application can checkpoint itself.

BLCR tries to make every process checkpointable.

Periodic checkpoints

Checkpoint the job at regular intervals.

On system startup, restart jobs from their last complete checkpoint.

Useful for systems with long jobs, fast I/O, and/or high node failure rates.

In normal processing, the periodic checkpoints are an expensive waste of time.

CIFTS

Coordinated Infrastructure for Fault Tolerant Systems

Parent project. Building a notification infrastructure for BLCR.

http://www.mcs.anl.gov/research/cifts/

Debugging

Roll back execution to a state saved before an error, and restart with a debugger.

Attaching a debugger to a restarted process.

- `cr_restart --stop`: restart a job in a suspended state.

- We used this heavily while porting BLCR to PPC64.
- Save your bugs (e.g. race conditions or issues that take hours to reproduce) for later inspection.

Implementation

BLCR provides single node checkpoint/restart through a kernel module and runtime library.

- *libcr.so: registers handlers, requests checkpoints*
- *libcr_run.so*: Stub library with a default checkpoint handler
- *blcr.ko*: coordinates the process checkpoints, saves (restores) kernel data structures, interfaces with library and command line tools.
- *blcr_insmod.ko*: provides kernel symbols

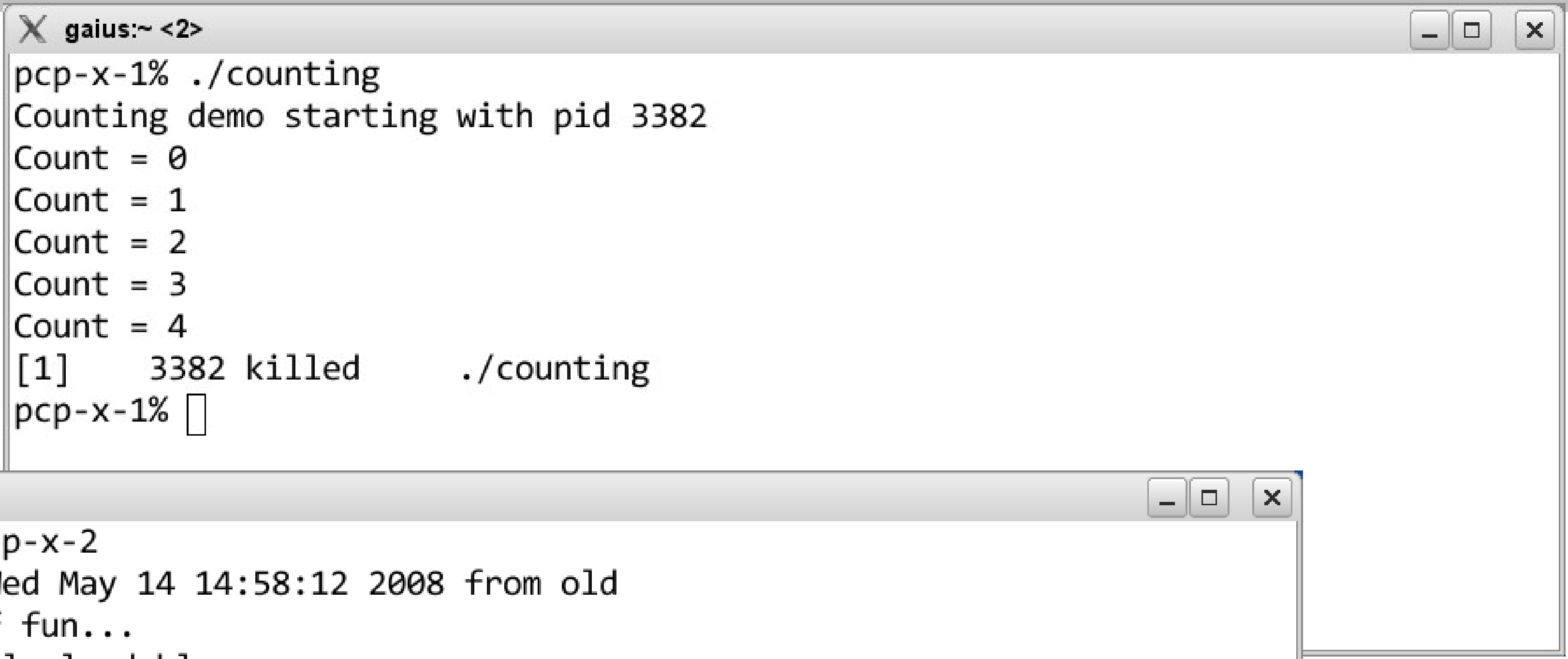
We don't support distributed operating system features

- No built-in support for TCP sockets, namespaces, etc.

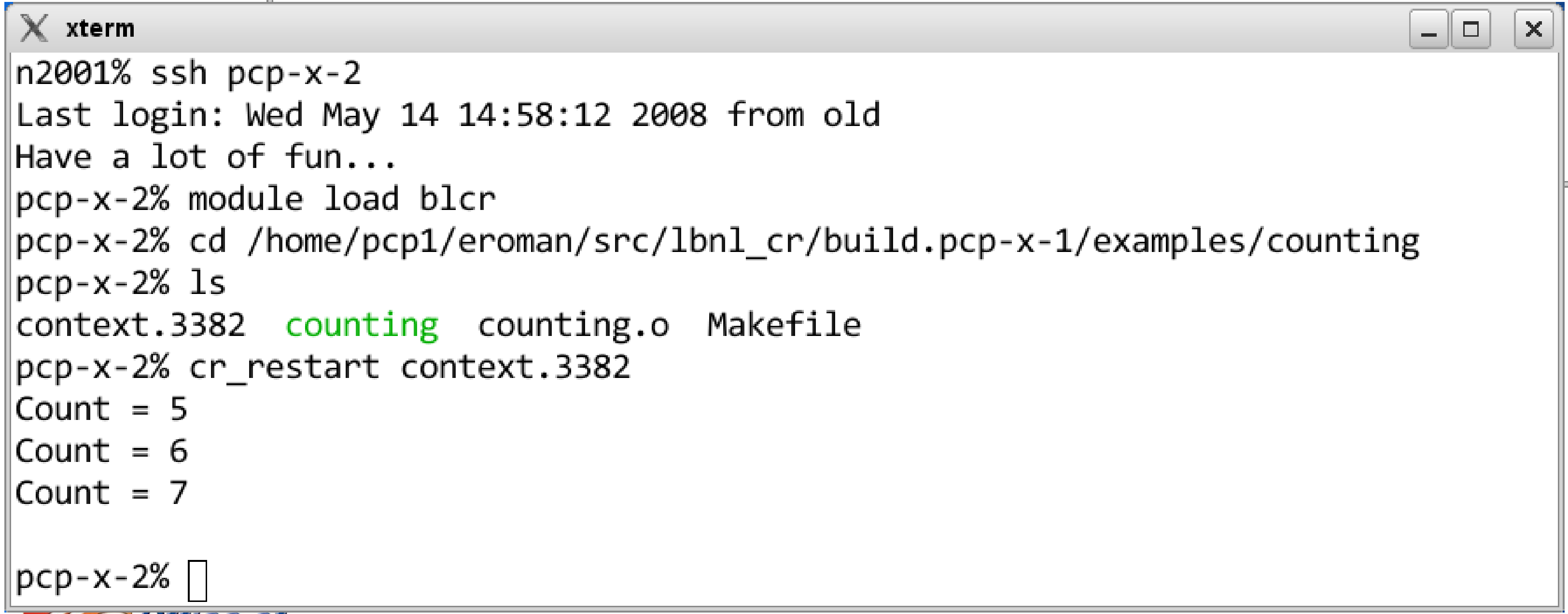
We provide the hooks to allow parallel runtimes/libraries to coordinate checkpoints and restart the processes through *handlers*.

The MPI library must know how to checkpoint; the user application does not.

Example: Migrating A Process



```
pcp-x-1% ./counting
Counting demo starting with pid 3382
Count = 0
Count = 1
Count = 2
Count = 3
Count = 4
[1]    3382 killed     ./counting
pcp-x-1%
```



```
n2001% ssh pcp-x-2
Last login: Wed May 14 14:58:12 2008 from old
Have a lot of fun...
pcp-x-2% module load blcr
pcp-x-2% cd /home/pcp1/eroman/src/lbnl_cr/build.pcp-x-1/examples/counting
pcp-x-2% ls
context.3382  counting  counting.o  Makefile
pcp-x-2% cr_restart context.3382
Count = 5
Count = 6
Count = 7

pcp-x-2%
```

Basic operation

Rough idea: Send the application a signal that tells it to call into BLCR.

$ cr_run counting (or use LD_PRELOAD)

call cr_init() (either directly or through libcr_run.so) to register default handler.

$ cr_checkpoint <pid-of-counting>

ioctl(CR_CHECKPOINT_REQUEST)

kernel sends signal to counting process and its children

the thread handlers in each child runs and invoke cr_checkpoint()

the signal handlers in each child runs and invoke cr_checkpoint()

once each child has called cr_checkpoint(), blcr.ko dumps all of the process state into a context file.

once all the checkpoints are complete, control returns to handlers.

the signal handlers finish, and control is returned to the application.

cr_checkpoint blocks waiting for all of this to happen.

Extending BLCR with Callbacks

Applications and libraries can use callbacks to save and restore unsupported objects, or receive notification of checkpoint, restart, and continue events.

Register callbacks as needed, usually at startup.

Callbacks run at checkpoint time, then resume at restart/continue

Can protect critical sections against callback execution.

BLCR interacts with the MPI library through callbacks.

BLCR coordinates within a node, handlers work between nodes.

Signal handler context

no thread-safety needed, but

Callbacks are limited to signal-safe functions (small subset of POSIX)

Thread context

Can call any function

Code needs to be thread-safe

Callback functions

```
my_callback {
   /* cr_checkpoint() returns twice. */
    ret = cr_checkpoint(0);

     if (ret > 0) {
         checkpoint_status = restart;
     } else if (ret == 0) {
         checkpoint_status = continue;
     } else {
         checkpoint_status = error;
     }

     return 0;
}
```

Status

Processes, process groups and sessions

- Shell scripts (bash, tcsh, python, perl, ruby, ...)
- Multithreaded processes (pthreads with standard NPTL)
- Resources shared between processes are restored.
- Restore PID and parent PID.

Files

- Reopen files during restart: open, truncate, and seek.
- Pipes and named FIFOs
- Files must exist in same location on filesystem
- Memory mapped files are remapped.
- New option to save shared libraries and executable.
- File path relocation

Supported Platforms

Linux kernel 2.6

- test with kernels from kernel.org, Fedora, SuSE, and Ubuntu
- support of custom patched kernels through autoconf

Architectures

- x86, x86-64, ppc, ppc64 and ARM
- Xen dom0 and domU

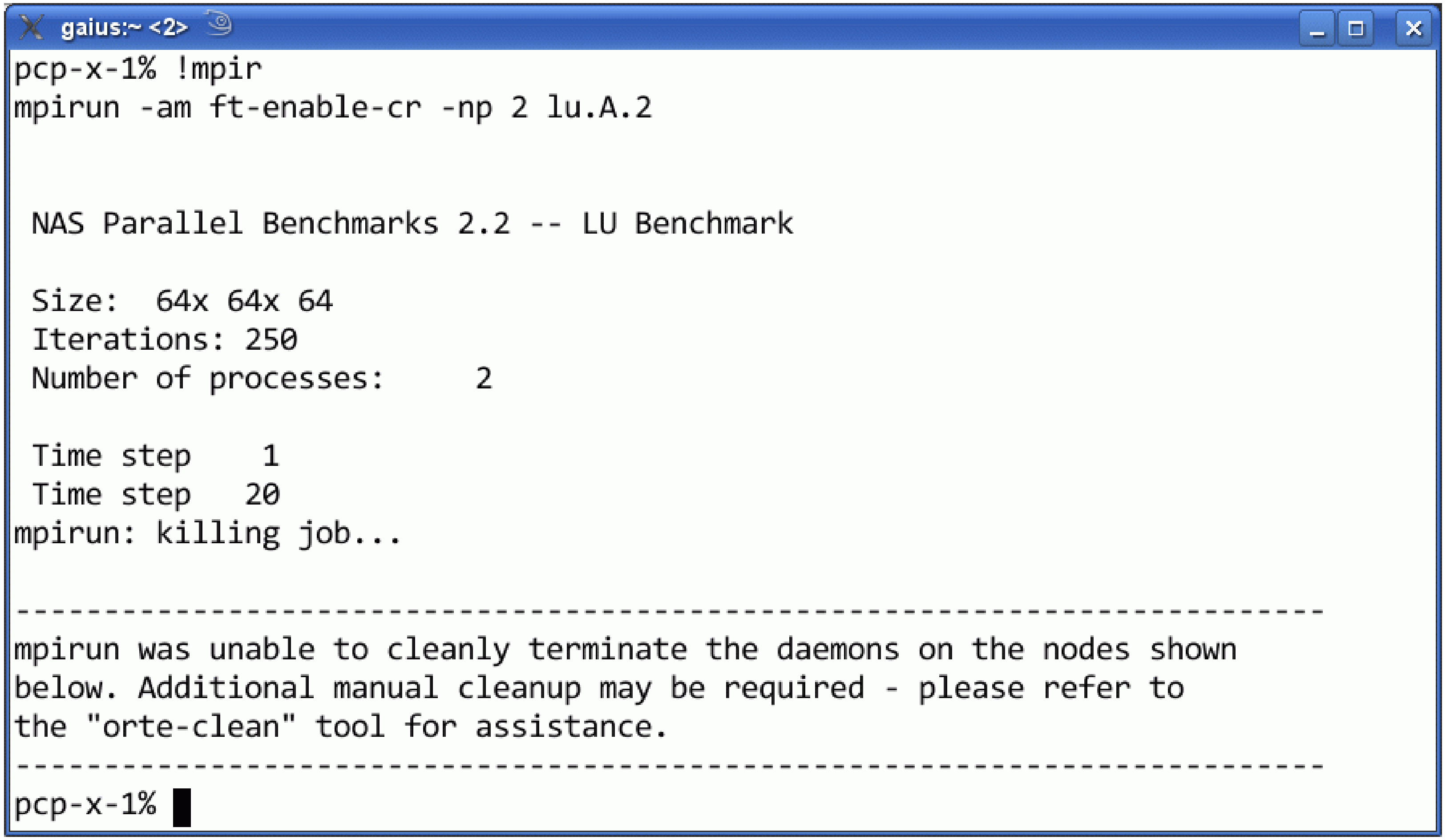
MPI

- MVAPICH2
- MPICH-V 1.0.x with sockets
- OpenMPI
- Cray Portals

Queue Systems

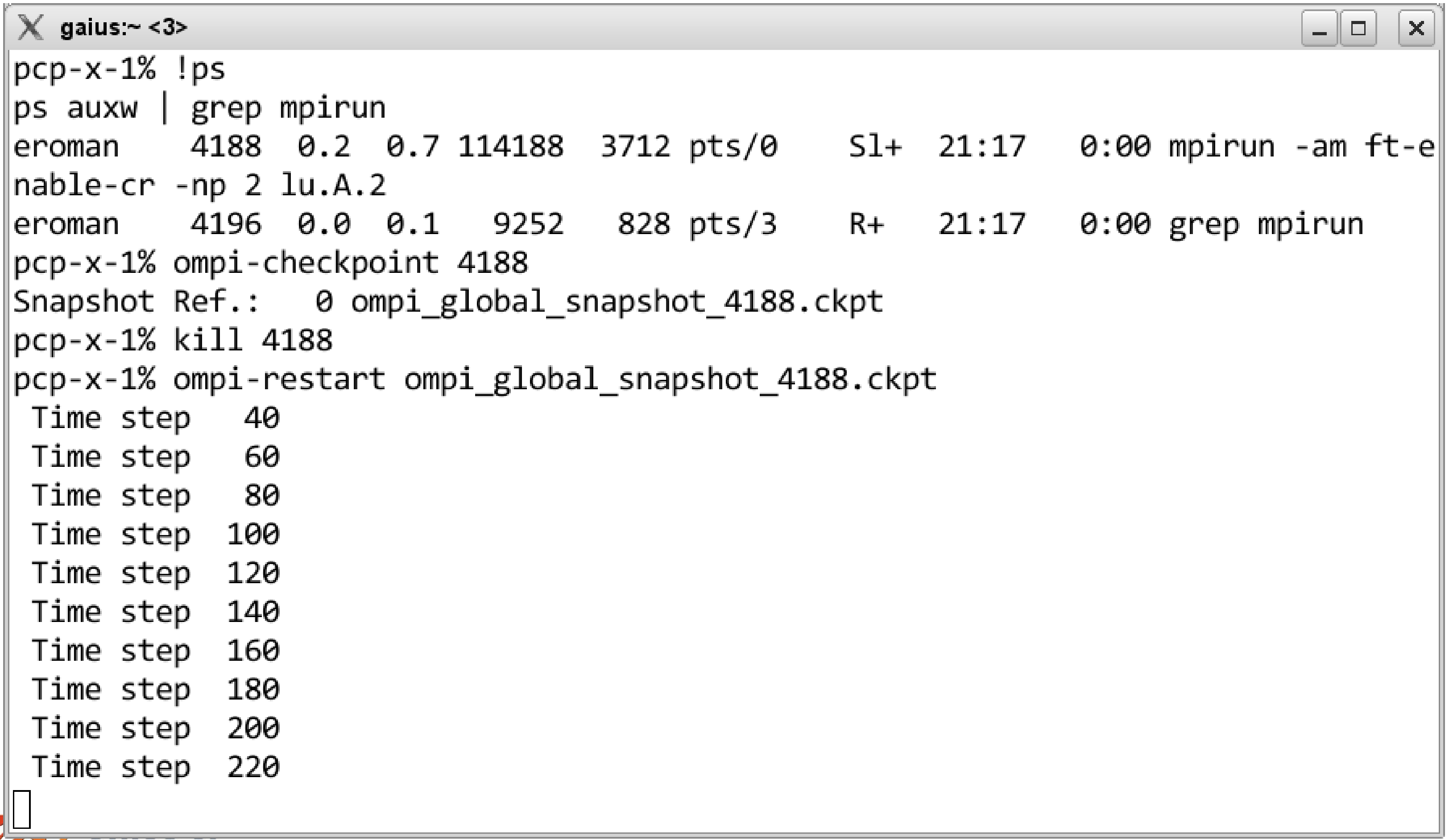
- Torque support available in recent snapshots.
- qhold, qrls, and periodic checkpoints tested.
- BLCR, Condor and Parrot HOWTO available.

Normal execution with Open MPI



```
pcp-x-1% !mpir
mpirun -am ft-enable-cr -np 2 lu.A.2



 NAS Parallel Benchmarks 2.2 -- LU Benchmark

 Size:  64x 64x 64
 Iterations: 250
 Number of processes:     2

 Time step    1
 Time step   20
mpirun: killing job...

--------------------------------------------------------------------------
mpirun was unable to cleanly terminate the daemons on the nodes shown
below. Additional manual cleanup may be required - please refer to
the "orte-clean" tool for assistance.
--------------------------------------------------------------------------
pcp-x-1% 
```

MPI: Checkpoint/Restart



gaius:~ <3>

```
pcp-x-1% !ps
ps auxw | grep mpirun
eroman    4188  0.2  0.7 114188  3712 pts/0    Sl+  21:17   0:00 mpirun -am ft-e
nable-cr -np 2 lu.A.2
eroman    4196  0.0  0.1   9252   828 pts/3    R+   21:17   0:00 grep mpirun
pcp-x-1% ompi-checkpoint 4188
Snapshot Ref.:   0 ompi_global_snapshot_4188.ckpt
pcp-x-1% kill 4188
pcp-x-1% ompi-restart ompi_global_snapshot_4188.ckpt
 Time step   40
 Time step   60
 Time step   80
 Time step  100
 Time step  120
 Time step  140
 Time step  160
 Time step  180
 Time step  200
 Time step  220
```

Work in progress

Queue system support

- BLCR, Torque, and OpenMPI
- Preemptive scheduling via priority queues under Maui
- Perodic checkpoints

Incremental checkpoints

- Collaboration with Frank Mueller at NCSU

Improved IO

- Runtime compression and decompression of context files

Conclusions

Future Work

Interested in other queue systems (LSF, SGE, SLURM, etc.)

More MPI implementations

MPICH 2 support anticipated

Torque support available.

You should be able to install BLCR on your system and checkpoint your MPI applications with it.

We would like you to download BLCR and try it!

For More Information

http://ftg.lbl.gov/checkpoint

Papers (available from website):

- "*Design and Implementation of BLCR*": high-level system design, including description of user API
- "*Requirements for Linux Checkpoint/Restart*": exhaustive list of Unix features we will support (or not).
- "*A Survey of Checkpoint/Restart Implementations*": focusing on open source versions that run on Linux
- "*The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing*": implementation with LAM/MPI

Other Approaches

Application-based checkpointing
- Efficient: save only needed data as step completes
- Good for fault tolerance: bad for preemption
- Requires per-application effort by programmer

Library-based checkpointing
- Portable across operating systems
- Transparent to application (but may require relink, etc.)
- Can't (generally) restore all resources (ex: process IDs)
- Can’t checkpoint shell scripts

Hypervisor (similar arguments for software suspend)
- Granularity is a full virtual machine
- Administrators have to maintain one VM per checkpoint
- Rollback. What happens to the disk state?
- Debugging?
- Coordination between multiple machines still necessary.
- Scheduler integration

Reactive checkpoints

If a node failure is imminent, checkpoint jobs affected by the failure.

Migrate the jobs away from the failing node, or wait for the node to be repaired.