CRD

BERKELEY LAB

Berkeley Lab's Checkpoint Restart (BLCR)

Eric Roman, Paul Hargrove, and Jason Duell
May 15, 2008

Office of Science
U.S. DEPARTMENT OF ENERGY

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?
Current status.	What does BLCR do now?
Plans.	Where is BLCR going?

Project Goals

Provide a checkpoint/restart facility for Linux clusters running scientific workloads.

Fit easily into production systems

Checkpoint and restart shell scripts running MPI applications.

No modifications to application source.

Use existing binaries. Users should not have to relink codes.

Runs on standard kernels as a kernel module. No kernel patches are necessary.

Support a wide variety of interconnects

Restart processes without runtime support. No C library (*libc)* modifications.

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

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

Scheduling

CR is a simple way to place a large parallel job into a waiting state.

- CR can be used to preempt running jobs.
- Drain queues quickly by checkpointing all of the running jobs. Restart those jobs after the system reboots.
- Prime time. Increase system throughput by checkpointing long jobs to make room for large parallel jobs.
- Increase system utilization by allowing the scheduler to correct for bad decisions.
- Gang scheduling. Divide system time up into slots.
- Priority scheduling. Run jobs with the highest priority.

Process migration.

- Pack jobs for optimal network performance.
- Move jobs to faster nodes.
- More scheduling flexibility if preemption is used.

Fault Tolerance

Checkpoint/restart can be used to implement process rollback.

- 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.

Automatic 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.

CIFTS

- Coordinated Infrastructure for Fault Tolerant Systems
- Parent project. Building a notification infrastructure for BLCR.
- http://www.mcs.anl.gov/research/cifts/

Debugging

Save application state at different times during execution.

Right now, you can attach the debugger to a restarted job.

- BLCR can restart a job in a suspended state.
- We used this heavily while porting BLCR to PPC64.
- Race conditions can become easy to reproduce.

We're interested in working with someone to add support for BLCR to gdb.

- The interfaces for checkpoint/restart are present in gdb.
- Parallel debugging?

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_vmadump.ko: saves the contents of virtual memory, registers, and related kernel data structures.

blcr_insmod.ko: provides kernel symbols

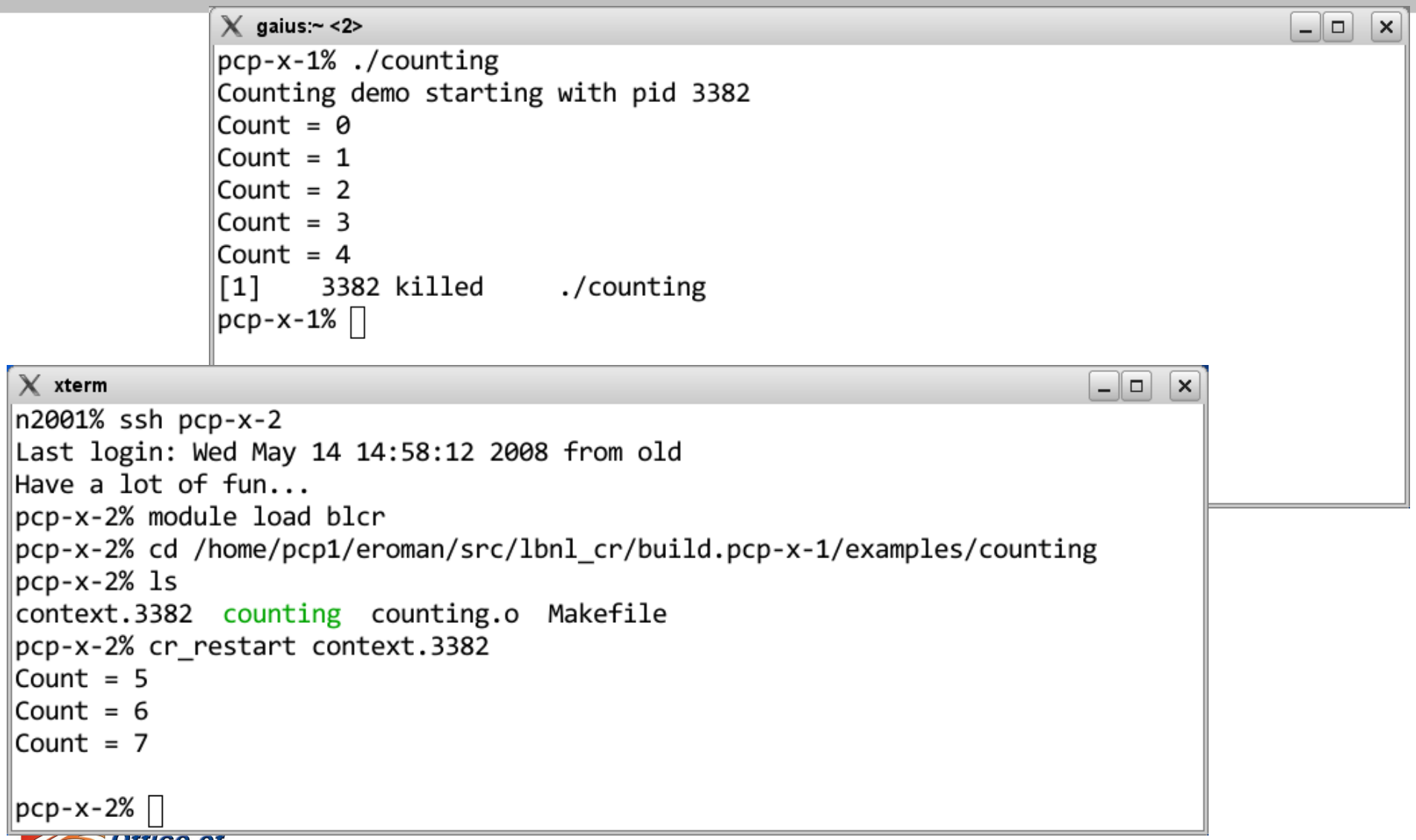
We don't support distributed operating system features

No built-in support for TCP sockets, bproc 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



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(), the 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.

Extension Interface

Callbacks

- Registered at startup (or as needed)
- Run at checkpoint time, then resume at restart/continue
- Save and restore unsupported objects
- Critical sections to protect uncheckpointable sections of code

- BLCR interacts with the MPI library through callbacks.
- BLCR coordinates within a node, handlers work between nodes.

Signal handler context

- Run with same PID (LinuxThreads); no thread-safety needed
- But callback limited to calling 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

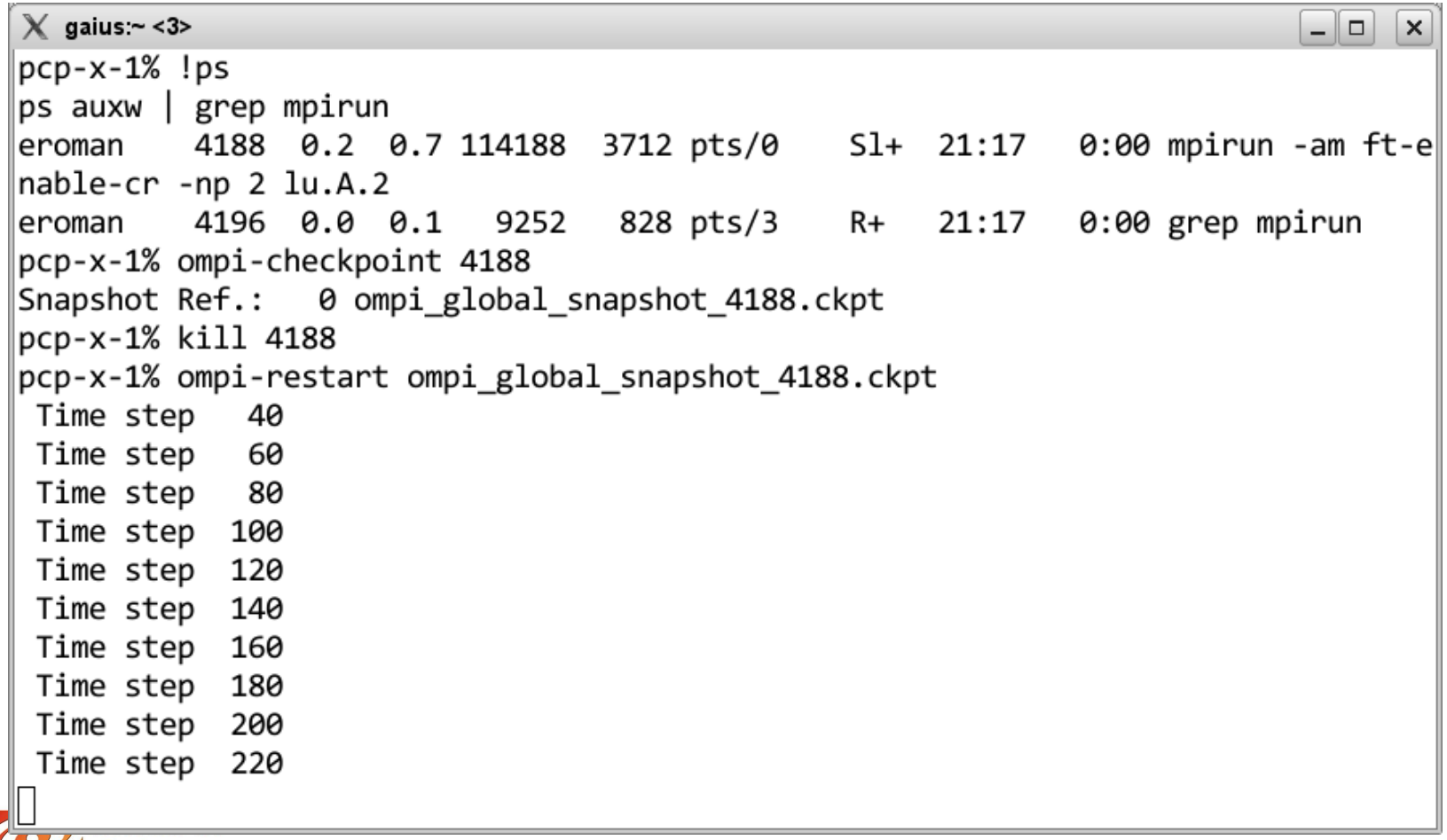
Supported Platforms

- Linux kernel 2.6 kernels
  - test with kernels from kernel.org, Fedora, SuSE, and Ubuntu
  - support of custom patched kernels through autoconf
  - 2.4 support is deprecated
- Architectures
  - x86, x86-64, ppc64 and ARM
  - Xen dom0 and domU
  - ppc32 planned.
- MPI
  - MVAPICH2
  - LAM/MPI 7.x (sockets and GM)
  - 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

```
gaius:~ <2>
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



Work in progress

Queue system support

- BLCR, Torque, and OpenMPI

Better semantics for files

- Allow checksum of file, with restart error if it has changed
- Allow saving contents of file (restore either clobbers, or opens anonymously)
- Support files that are not open at checkpoint time, but are specified as being part of the checkpoint

Improved IO

- On-the-fly compression of context files
- Direct IO

Robust error reporting

Zombie processes

Conclusions

Future Work

- Interested in other queue systems (LSF, SGE, SLURM, etc.)
- More MPI implementations
- MPICH 2 support anticipated
- Vendor support (Quadrics)?
- LAM/MPI support for partial/live migration
- Ship support with distributions (ROCKS, OSCAR)

We expect BLCR to be deployed in a production batch environment before the end of the calendar year.

Torque support will be available soon.

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