CRD

BERKELEY LAB

System-level Checkpoint/Restart with BLCR

Paul Hargrove
(work with Eric Roman and Jason Duell)
checkpoint@lbl.gov

LACSS 2008

October 15, 2008 Santa Fe, New Mexico

Office of Science
U.S. DEPARTMENT OF ENERGY

Introduction

Checkpoint.	Save a process's state to a file descriptor.
Restart.	Reconstruct the process from a file descriptor.
BLCR.	**B**erkeley **L**ab **C**heckpoint **R**estart 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 checkpoint/restart for Linux clusters running scientific workloads

Checkpoint and restart *jobs* (shell scripts) running MPI applications.

Support a wide variety of networks.

Fit easily into production systems

Run *unmodified* application source.

Run *unmodified* binaries where possible. No special compile/link in most cases.

Run on *unpatched* kernels (as a kernel module).

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.

Usage Scenarios

Batch Scheduling.

- C/R can be used to pre-empt and/or migrate running jobs.
- Drain queues quickly for maintenance.
- Increase system throughput by switching job mix between long jobs and wide 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.

Fault Tolerance.

- Not every application can checkpoint itself.
- Periodic checkpoints can reduce lost work in case of failure (but adds cost to normal fault-free execution).
- Reactive checkpoints can respond to non-yet-fatal problems (like loss of a fan).

Debugging.

- Rollback execution to a checkpoint taken before a fault, restart with a debugger.

Other Approaches

Application-based checkpointing.

- Efficient: save only needed data as step completes.
- Good for fault tolerance: bad for preemptive scheduling.
- 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 for distributed jobs is still necessary.
- Scheduler integration.

Implementation

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

libcr.so: Full library: can register handlers, request checkpoints, etc.

OR *libcr_run.so*: Stub library with only a default checkpoint handler

Kernel modules: coordinates the process checkpoints, saves/restores kernel data structures, interfaces with library and command line tools.

BLCR doesn't provide built-in support for distributed runtime features

TCP sockets, bproc namespaces, etc.

Instead, BLCR provides hooks which allow apps and libraries to coordinate checkpoints and restart distributed processes through *callbacks*.

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

Basic Operation

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

A checkpoint request can come from the same process, or from another.

By default, user code doesn't *need* to do anything to handle it.

If desired, user code *may* register a callback to handle it.

If desired, user code *may* block requests (critical sections).

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 (incl. shared libs and executable).
- Option to save shared libraries and executable.
- Option for 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.

Architectures

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

MPI Implementations

- MVAPICH2
- LAM/MPI 7.x (sockets and GM)
- MPICH-V 1.0.x with sockets
- OpenMPI
- Cray Portals

Batch Queue Systems

- TORQUE support
  - Available in recent releases.
  - Have tested qhold, qrls, and periodic checkpoints.
- BLCR, Condor and Parrot HOWTO available.

Example 2: MPI Checkpoint/Restart

Step 1 (mpirun) and Step 2 (checkpoint)

```
gaius:~ <2>
pcp-x-1% 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
```

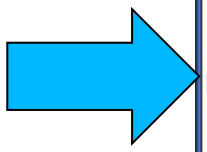
1

2

```
gaius:~ <3>
pcp-x-1% ps aux | grep mpirun
eroman    4188  0.2  0.7 114188  3712 pts/0    Sl+  21:17   0:00 mpirun -am ft-e
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
```

Example 2: MPI Checkpoint/Restart

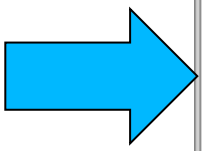
The job terminates...



```
gaius:~ <2>
pcp-x-1% 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%
```

Example 2: MPI Checkpoint/Restart

The job restarts...



```
gaius:~ <3>
pcp-x-1% ps aux | grep mpirun
eroman    4188  0.2  0.7 114188  3712 pts/0    Sl+  21:17   0:00 mpirun -am ft-e
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 w/ TORQUE + OpenMPI (they work individually).

Alternative handling of files

Allow checksum of file, with restart error if it has changed.

Allow saving contents of files (restore may either replace or rename).

Support files that are not open at checkpoint time, but are specified as by the checkpoint requester.

Improved I/O

On-the-fly compression of context files.

Direct I/O.

Other

Detailed error reporting (*e.g.* What file caused ENOENT?)

Zombie processes.

Future Work

Future Work

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

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

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

Addressing the I/O Bottleneck

Concern

- MTBF is shrinking.
- Time to complete global checkpoint is growing (I/O dominates).
- When they meet checkpoint/restart is no longer viable.

Some ideas we are working on

- Compression: we find highly app-dependent results (mixed).
- Incremental: use page tables to identify modified memory.
- Differential: rsync-type mechanism to identify modified memory.
- Memory exclusion: allow user or compiler to exclude dead memory.

Some ideas for others

- More intelligent (non-global) checkpointing algorithms.
- Checkpoint to local storage (low-cost solid state devices).
- Checkpoint to memory or storage of near-by node.
- Reactive in place of periodic as normal case.

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

CIFTS

Coordinated Infrastructure for Fault Tolerant Systems

Parent project. Subject of the 11am talk by Rinku Gupta.

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