Cornell University
Center for Advanced Computing

Autosave for Research
Where to Start with Checkpoint/Restart

Brandon Barker
Computational Scientist
Cornell University Center for Advanced Computing (CAC)
brandon.barker@cornell.edu

Workshop: High Performance Computing on Stampede

January 15, 2015

The problem

1. You test out your newly developed software on a small dataset.
2. All is well, you submit a big job and go read some papers or watch some TV.
3. Several days later, one of the following happens:
   - Someone uses up all the memory on the system.
   - Power failure
   - Unplanned maintenance
   - ???

Ad-hoc solutions

i.e. dodgy, incomplete, and error-prone solutions

- Save data every N iterations
  - Takes time to code.
  - May miss some data.
  - Have to write custom resume code.
  - For all of these, different sections of the program may need different *save* and *restore* procedures.
- For some tasks: run on discrete chunks of data.
  - Works best when
    - There are many data items.
    - Each item can be processed quickly.
    - There are no dependencies between items
  - In short, embarrassingly parallel programs can use simple book-keeping for C/R.
  - Still, it involves some work on the part of the researcher for restoration.

What is C/R?

Think of virtual machines: if you've ever saved and restarted a virtual machine or emulator, you have used a type of C/R!

Checkpoint: save the program state

- Program memory, open file descriptors, open sockets, process ids (PIDs), UNIX pipes, shared memory segments, etc.
- For distributed processes, need to coordinate checkpointing across processes.

Restart: restart the process with saved state

- Some of the above require special permissions to restore (e.g. PIDs); not all C/R models can accommodate this. Others (like VMs) get it for free.
- Some of the above may be impossible to restore in certain contexts (e.g. sockets that have closed and cannot be re-established).

Use cases of C/R

- Recovery/fault tolerance (restart after a crash).
- Save scientific interactive session: R, MATLAB, IPython, etc.
- Skip long initialization times.
- Interact with and analyze results of in-progress CPU-intensive process.
- Debugging
  - Checkpoint image for ultimate in reproducibility.
  - Make an existing debugger reversible
- Migrate process (or even multiple VMs).
- Robust “exception handling” in languages without exceptions or garbage collectors.

Virtual Machine (VM) C/R

VM-level C/R is relatively easy to implement, once you have a VM: the system is already isolated.

Implementations

- Most any hypervisor platform: KVM, Virtualbox, VMWare, etc.
  - KVM is relatively lightweight.

Pros

- Very simple to use.
- Few surprises.
- Many applications supported; few limitations.

Cons

- Operating in a VM context requires predefined partitioning of RAM and CPU resources.
- More overhead in most categories (storage of VM image, RAM snapshot, etc.).
- Still a challenge for multi-VM C/R.

Containers with C/R

Containers are a form of virtualization that uses a single OS kernel to run multiple, seemingly isolated, OS environments.

Implementations

- OpenVZ
- CRIU – C/R In Userspace
- (Not all containers support C/R)

Pros

- Like VMs, enjoy the benefit of existing virtualization technology.
- *Fewer* surprises.

Cons

- May incur additional overhead, due to C/R of unnecessary processes and storage.
- Still a challenge for multi-VM C/R.

Kernel-modifying C/R

Requires kernel modules or kernel patches to run.

Implementations

- OpenVZ
- BLCR – Berkeley Lab C/R
- ~~CRIU~~ – But now in mainline!

Pros

- Varied.

Cons

- Required modification of the kernel.
- May not work for all kernels (BLCR does not past 3.7.1).

(Multi) application C/R

Checkpoint one or several interacting processes. Does not use the full container model.

Implementations

- BLCR
- CRIU
- DMTCP – Distributed MultiThreaded CheckPointing

Pros

- Usually simple to use.
- Lower overhead.

Cons

- May have surprises; applications use different advanced feature sets (e.g. IPC), and each package will have a different feature set. **Test first!**
- BLCR requires modification of application for static linking.
- DMTCP static linking support is experimental
- CRIU is a bit new.

Custom C/R

This is like ad-hoc, but when you do it even though you know other C/R solutions exist.

Libraries that help

- (p)HDF5
- NetCDF

Pros

- Very low over-head.
- Few surprises if done properly.

Cons

- Needs thorough testing for each application.
- Lots of development time.
- Less standardization.
- Always a chance something is missed.

What is a good C/R solution for HPC?

Requirements

- Must be non-invasive
  - No kernel modifications
  - Preferably no libraries needed on nodes.
- Should have low overhead.
- Must support distributed applications.

Bonuses

- Easy to use.
- Stable for the user.

It looks like DMTCP is the best candidate, for now.

An overview of DMTCP

- Distributed MultiThreaded CheckPointing.
  - Threads (OpenMP, POSIX threads), MPI.
- Easy to build and install library.
- Not necessary to link with existing dynamically linked applications.
  - DMTCP libs replace (wrap) standard libs and syscalls.
  - DMTCP lib directory should be in LD_LIBRARY_PATH and LD_PRELOAD (handled by DMTCP scripts).

Counting in C

```
#include <stdio.h>
#include <unistd.h>

int main(void) {
  unsigned long i = 0;
  while (1) {
    printf("%lu ", i);
    i = i + 1;
    sleep(1);
    fflush(stdout);
  }
}
```

- dmtcp_checkpoint -i 5 ./count
- dmtcp_restart ckpt_count xxx.dmtcp

Counting in Perl

```
#/usr/bin/perl -w

$| = 1; # autoflush STDOUT

$i = 0;

while (true) {
  print "$i ";
  $i = $i + 1;
  sleep(1);
}
```

- dmtcp_checkpoint -i 5 perl count.pl
- dmtcp_restart ckpt_perl_xxx.dmtcp

X11 (graphics) support

- All current non-VM C/R relies on VNC for X11 support.
- DMTCP has a known bug with checkpointing xterm.
- Due to dependence on VNC and general complications, only try to use if you have to.
- Not supported on Stampede or most other large HPC systems.

Reversible Debugging with FReD

- Supports GDB and several interpreters
- Allows you to inspect one part of the program, then go back to a previous state without restarting the debugger.
- github.com/fred-dbg/fred
- youtu.be/1l_wGZz0JEE

DMTCP plugins

- Used to modify the behavior of DMTCP
  - Modify behavior at the time of checkpoint or restart.
  - Add wrapper functions around library functions (including syscalls).
  - Much of DMTCP itself is now written as plugins.
  - Custom plugins could add support for callbacks in user’s program.

Conclusions

- DMTCP appears to be good for most jobs for now. Also easy to install.
- CRIU will likely be a strong contender in the future, but it is not yet ready for HPC.
- BLCR and OpenVZ may be more robust than DMTCP for PID restoration (provided your kernel has support for BLCR or OpenVZ).
- For the foreseeable future, it is unlikely that any one C/R framework will meet everyone's needs.

Additional Resources

- Source and other docs: github.com/cornell-comp-internal/CR-demos
- FReD: github.com/fred-dbg/fred
- Python and DMTCP: youtu.be/1l_wGZz0JEE
- Comparison Chart: criu.org/Comparison_to_other_CR_projects