damien.francois@uclouvain.be
UCL/CISM

An introduction to checkpointing for scientific applications

November 2017
CISM/CÉCI training session

What is checkpointing

```
$ ./count
```

```
$ ./count
1
```

```
$ ./count
1
2
```

```
$ ./count
1
2
3
```

```
$ ./count
1
2
3^C
$
```

```
$ ./count
1
2
3^C
$ ./count
```

```
$ ./count
1
2
3^C
$ ./count
1
```

Without checkpointing:

```
$ ./count
1
2
3^C
$ ./count
1
```

Without checkpointing:

```
$ ./count
1
2
3^C
$ ./count
1
```

With checkpointing:

```
$ ./count
1
2
3^C
$ ./count
4
```

Without checkpointing:

```
$ ./count
1
2
3^C
$ ./count
1
2
```

With checkpointing:

```
$ ./count
1
2
3^C
$ ./count
4
5
```

Without checkpointing:

```
$ ./count
1
2
3^C
$ ./count
1
2
3
```

With checkpointing:

```
$ ./count
1
2
3^C
$ ./count
4
5
6
```

Without checkpointing:

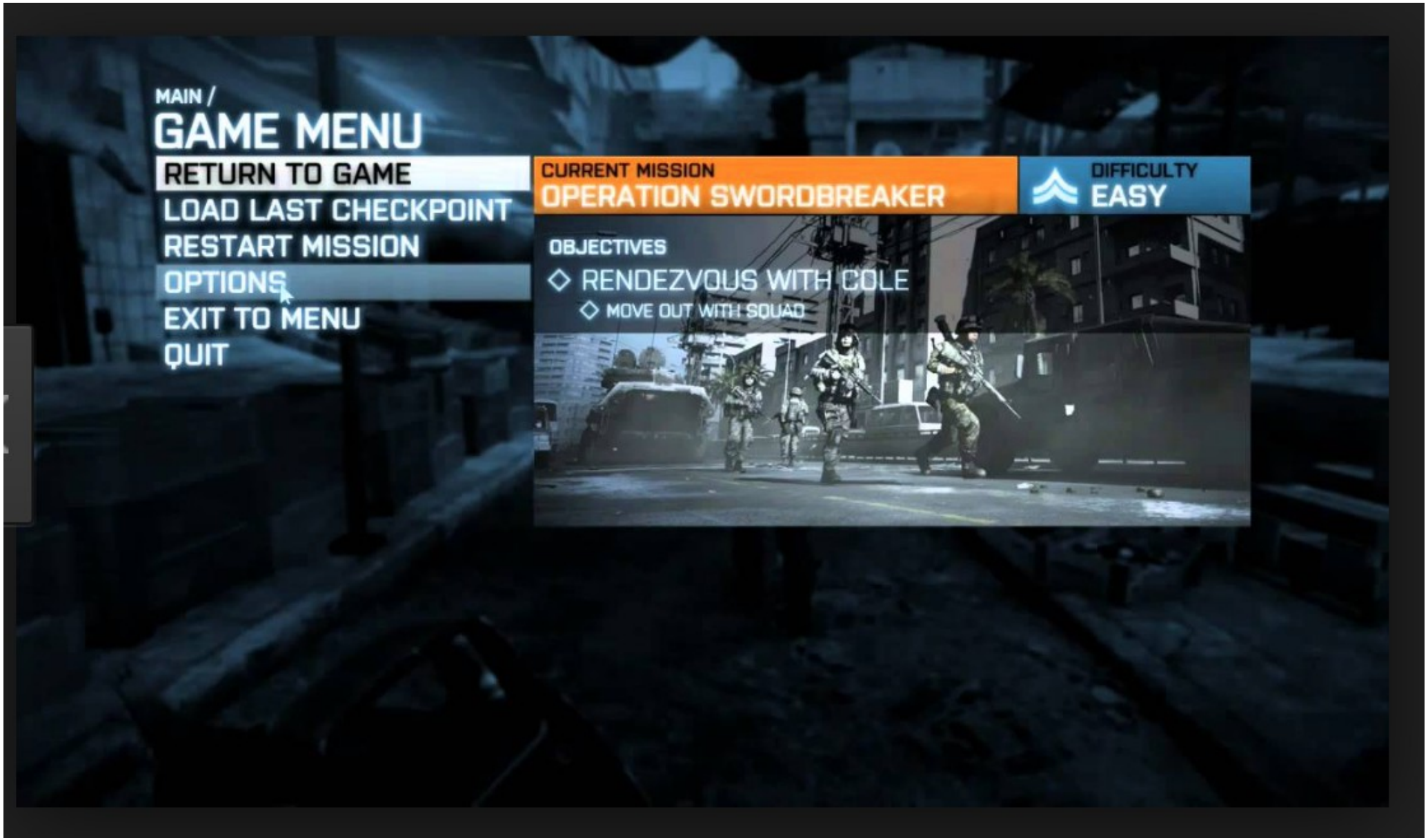
```
$ ./count
1
2
3^C
$ ./count
1
2
3
```

With checkpointing:

```
$ ./count
1
2
3^C
$ ./count
4
5
6
```

Checkpointing:

'saving' a computation
so that it can be resumed later
(rather than started again)


MAIN /
GAME MENU
RETURN TO GAME
LOAD LAST CHECKPOINT
RESTART MISSION
OPTIONS
EXIT TO MENU
QUIT
CURRENT MISSION
OPERATION SWORDBREAKER
DIFFICULTY
EASY
OBJECTIVES
RENDEZVOUS WITH COLE
MOVE OUT WITH SQUAD

The idea:

Save the program state

every time a checkpoint is encountered

and restart from there upon (un)planned stop

rather than bootstrap again from scratch

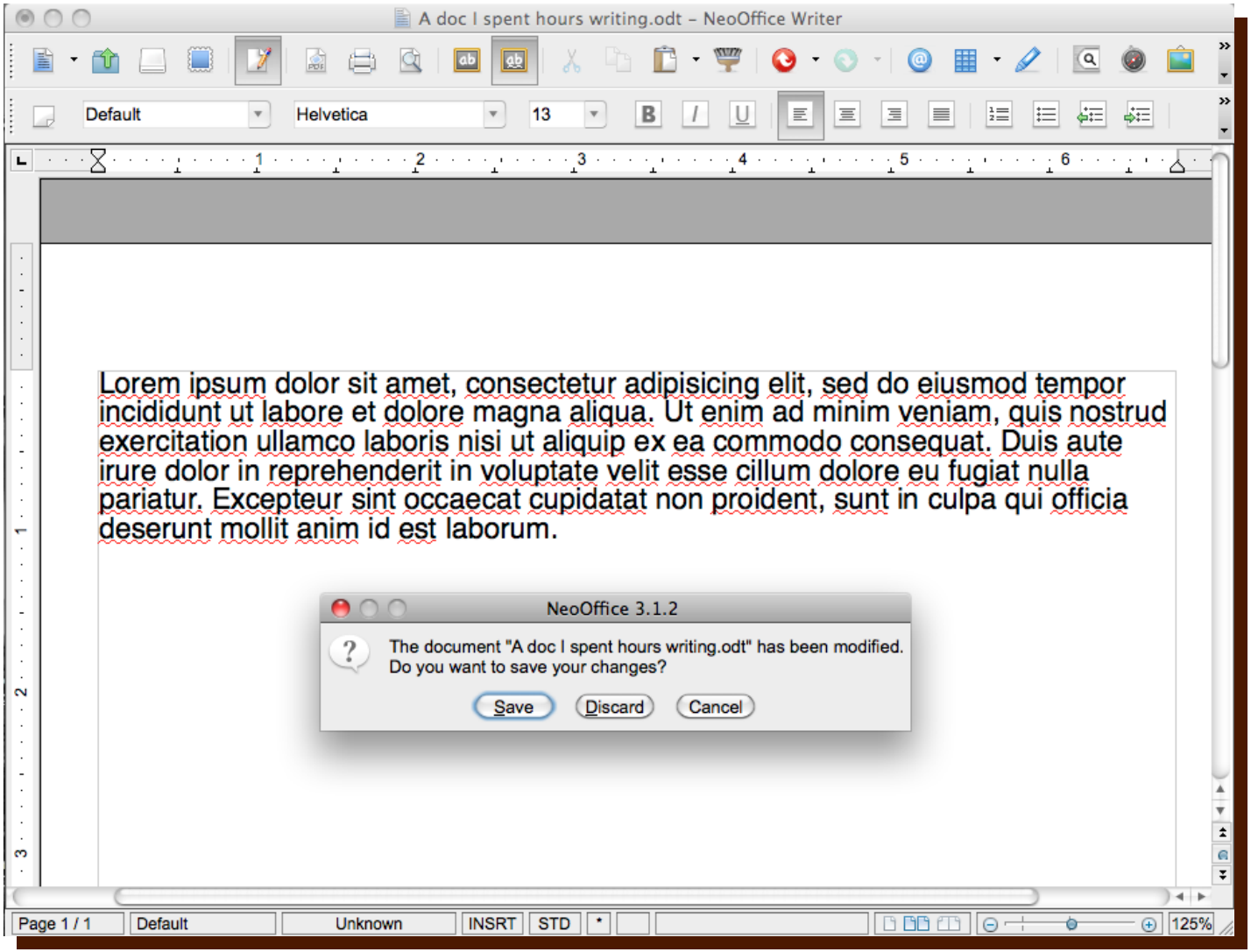
Values in variables
Open files
...

Position in the code
Signal or event
...

starting loops at iteration 0
creating tmp files
...

Why do we need checkpointing

Imagine a text editor without 'checkpointing' ...



Goals of checkpointing in HPC:

1. Fit in time constraints
2. Debugging, monitoring
3. Cope with hardware failures
4. Job preemption

How do we do checkpointing

Software level vs system level

Checkpointing at the **software** level

The software you use has checkpointing built-in	No	Yes
You are the author	You have some work to do.	Good job!
You are just a user	Keep listening...	Read the manual

Checkpointing at the **system** level

Software level vs system level

Characteristics	System Level	Application level
Triggered by:	User/system	Application
Basic idea	Full memory dump	Save relevant information
When to checkpoint?	Any time	Pre-fixed places
Requires modification of application	No (some technologies require re-compilation)	Yes
Resulting file size	Big	Small
Overhead in exec. time	~1-2%	negligible

1 Software with built-in checkpointing.

Working with checkpoint-restart-able software

Gaussian 09 Frequently Asked Question

How can I restart a job that was interrupted?

Many Gaussian jobs that are stopped prematurely — e.g., due to a machine crash, a power failure, manually killing the job — can be restarted. These include geometry optimizations, frequency calculations, and CCSD and EOM-CCSD calculations. The technique to restart the jobs varies depending on the type of job. This FAQ will discuss some common cases.

Be aware that all restarts require the checkpoint file from the previous job. Some job types also require the read-write file. If the required file(s) have been deleted, then the job cannot be restarted.

http://www.gaussian.com/g_blog/faq2.htm

Working with checkpoint-restart-able software

CFD Direct

The Architects of OpenFOAM

Search

Home OpenFOAM Cloud Training Support About Jobs

OpenFOAM User Guide: 4.3 Time and data input/output control

[Table of Contents][Index][Version 4 | Version 3]

[prev] [next]

4.3 Time and data input/output control

The OpenFOAM solvers begin all runs by setting up a database. The database controls I/O and, since output of data is usually requested at intervals of time during the run, time is an inextricable part of the database. The `controlDict` dictionary sets input parameters `essential` for the creation of the database. The keyword entries in `controlDict` are listed in the following sections. Only the time control and `writeInterval` entries are mandatory, with the database using default values for any of the optional entries that are omitted. Example entries from a `controlDict` dictionary are given below:

```

application     icoFoam;

startFrom       startTime;

startTime       0;
```

OpenFOAM Training

17 Oct Chicago, USA

07 Nov Munich, Germany

31 Jan Virtual, Americas

06 Feb London, UK

27 Feb Houston, USA

21 Mar Virtual, Europe

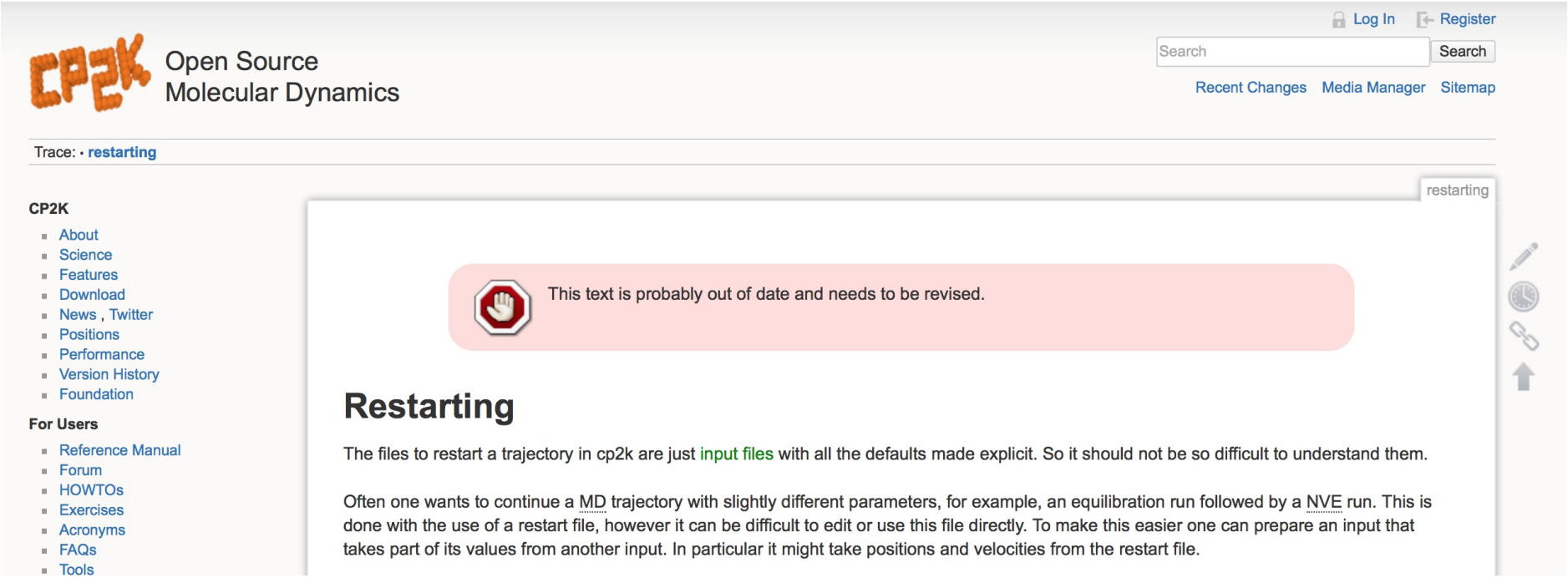
24 Apr Berlin, Germany

Recent Posts

Issues with OpenFOAM, Pt. 2

http://cfd.direct/openfoam/user-guide/controlDict/

Working with checkpoint-restart-able software



https://www.cp2k.org/restarting

Working with checkpoint-restart-able software

manual quickstart instguide update basis

Next: 19.3 Variables **Up:** 19 Advanced features of **Previous:** 19.1 Memory control **Contents**

19.2 Restarting calculations

By default, and in all examples shown so far, scratch files are used to store all intermediate data MOLPRO needs, and the user will normally not see these files at all. However, it is possible to save computed data as orbitals and energies in named (permanent) files and use these for restarting a calculation at a later stage. MOLPRO uses a number of different files, but only one or two of them are needed for a restart. File 1 holds the one- and two electron integrals and related information, while on file 2 the wavefunction information like orbitals, orbital energies, and optionally CI-vectors are stored. Thus, file 2 is essential for restarting a calculation, while the integrals on file 1 can either be restarted or recomputed.

Molpro Quantum Chemistry Software

https://www.molpro.net/info/2015.1/doc/quickstart/node65.html

Working with checkpoint-restart-able software

Next Up Previous Contents Index

Next: ICHARG-tag **Up:** The INCAR File **Previous:** MAGMOM-tag **Contents** **Index**

ISTART-tag

ISTART= 0 | 1 | 2

Default:

ISTART = 1 if WAVECAR exists
= 0 else

This flag determines whether to read the file WAVECAR or not.

0
Start job: begin 'from scratch'. Initialize the orbitals according to the flag INIWAV .

1
``restart with constant energy cut-off''. Continuation job -- read orbitals from file WAVECAR (usage is restricted in the parallel version, see section 4.5).

http://cms.mpi.univie.ac.at/vasp/vasp/ISTART_tag.html

Working with checkpoint-restart-able software

LAMMPS

Search docs

USER DOCUMENTATION

1. Introduction
2. Getting Started
3. Commands
4. Packages
5. Accelerating LAMMPS performance
6. How-to discussions
7. Example problems
8. Performance & scalability

Docs » Commands » restart command

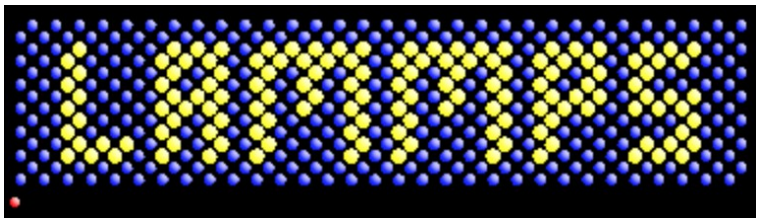
Website Commands

Previous Next

restart command

Syntax

```
restart 0
restart N root keyword value ...
restart N file1 file2 keyword value ...
```



http://lammps.sandia.gov/doc/restart.html

Working with checkpoint-restart-able software

GROMACS FAST. FLEXIBLE. FREE.

Log in Register

Search Search

Main pages

- Gromacs
- Documentation
- How-tos
  - Adding a Residue to ...
  - Analysing Trajectory...
  - Beginners
  - Carbon Nanotube
  - Changing a 3 Point ...

Doing Restarts

General

To achieve an exact restart of a simulation, one must preserve all the state variables of the system. In practice, this translates into preserving coordinates, velocities, and energy components in high precision. Most of the discussion below addresses how to restart a crashed simulation in GROMACS 3.x. GROMACS 4.x is much simpler and can be dealt with first.

Version 4.x (and later)

With the introduction of checkpointing, the instructions given below for 3.x are partly obsolete. They should still work, but the simple cases of breaking up a long **mdrun** or recovering from a crash are now easier. If a simulation crashes, make use of the **state.cpt** file that is written; it contains all of the information necessary to continue the simulation. In order to pick up from where the simulation stopped, simply use the **-cpi** and **-append** options to mdrun. Note that -append is the default in 4.5.

Before doing anything, back up your files! Then use

```
mdrun -s topol.tpr -cpi state.cpt
```

http://www.gromacs.org/Documentation/How-tos/Doing_Restarts

Working with checkpoint-restart-able software

Go to the top | Complete list of input variables

restartxf

Mnemonics: RESTART from (X,F) history

Characteristic:

Variable type: integer parameter

Default is 0.

Control the restart of a molecular dynamics or structural optimization job.

restartxf>0 (Deprecated):The code reads from the input wf file, the previous history of atomic coordinates and corresponding forces, in order to continue the work done by the job that produced this wf file. If optcell/=0, the history of acell and rprim variables is also taken into account. The code will take into consideration the whole history (if **restartxf**=1), or discard the few first (x,f) pairs, and begin only at the pair whose number corresponds to **restartxf**.

Works only for ionmov=2 (Broyden) and when an input wavefunction file is specified, thanks to the appropriate values of irdwfk or getwfk.

http://www.abinit.org/doc/helpfiles/for-v7.10/input_variables/varrlx.html#restartxf

2 Software you write yourself

The general recipe

1. Look for a state file
   (name can be hardcoded, or, better, passed as parameter)
2. If found, then restore state
   (initialize all variables with content of the file state)

   Else, bootstrap (create initial state)
3. Periodically save the state

1. dfr@manneback (ssh)

```c
// gcc count.c -o count && ./count
#include <stdio.h>
void main()
{
  int i, the_start, the_end;

  the_start = 1;
  the_end = 10;

  for (i=the_start; i<=the_end; i++)
  {
    printf("%d\n", i);
    sleep(1);
  }
}
```

count.c 12,5 All

```c
// gcc crcount.c -o crcount && ./crcount
#include <stdio.h>
void main()
{
  int i, the_start, the_end;
  FILE * file;

  // Try to recover current state
  file = fopen("state", "r");
  if (file)
  {
    fscanf(file, "%d", &the_start);
    fclose(file);
  }
  else
  {
    // Otherwise bootstrap at 1
    the_start = 1;
  }

  the_end = 10;

  for (i=the_start; i<=the_end; i++)
  {
    // Save current state
    file = fopen("state", "w");
    fprintf(file, "%d", i);
    fclose(file);

    // Heavy computations
    printf("%d\n", i);
    sleep(1);
```

crcount.c 3,1 Top

C recipe

UNIX processes can receive 'signals' from the user, the OS, or another process

SIGHUP	1	Exit	Hangup
SIGINT	2	Exit	Interrupt
SIGQUIT	3	Core	Quit
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace/Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status
SIGPWR	19	Ignore	Power Fail/Restart
SIGWINCH	20	Ignore	Window Size Change
SIGURG	21	Ignore	Urgent Socket Condition
SIGPOLL	22	Ignore	Socket I/O Possible
SIGSTOP	23	Stop	Stopped (signal)
SIGTSTP	24	Stop	Stopped (user)
SIGCONT	25	Ignore	Continued
SIGTTIN	26	Stop	Stopped (tty input)
SIGTTOU	27	Stop	Stopped (tty output)
SIGVTALRM	28	Exit	Virtual Timer Expired
SIGPROF	29	Exit	Profiling Timer Expired
SIGXCPU	30	Core	CPU time limit exceeded
SIGXFSZ	31	Core	File size limit exceeded
SIGWAITING	32	Ignore	All LWPs blocked
SIGLWP	33	Ignore	Virtual Interprocessor Interrupt for Threads Library
SIGAIO	34	Ignore	Asynchronous I/O

UNIX processes can receive 'signals' from the user, the OS, or another process

^C

^D

^Z

SIGHUP	1	Exit	Hangup
SIGINT	2	Exit	Interrupt
SIGQUIT	3	Core	Quit
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace/Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status
SIGPWR	19	Ignore	Power Fail/Restart
SIGWINCH	20	Ignore	Window Size Change
SIGURG	21	Ignore	Urgent Socket Condition
SIGPOLL	22	Ignore	Socket I/O Possible
SIGSTOP	23	Stop	Stopped (signal)
SIGTSTP	24	Stop	Stopped (user)
SIGCONT	25	Ignore	Continued
SIGTTIN	26	Stop	Stopped (tty input)
SIGTTOU	27	Stop	Stopped (tty output)
SIGVTALRM	28	Exit	Virtual Timer Expired
SIGPROF	29	Exit	Profiling Timer Expired
SIGXCPU	30	Core	CPU time limit exceeded
SIGXFSZ	31	Core	File size limit exceeded
SIGWAITING	32	Ignore	All LWPs blocked
SIGLWP	33	Ignore	Virtual Interprocessor Interrupt for Threads Library
SIGAIO	34	Ignore	Asynchronous I/O

`kill -9`

`kill`

`fg, bg`

UNIX processes can receive 'signals' from the user, the OS, or another process

e.g.

SIGHUP	1	Exit	Hangup
SIGINT	2	Exit	Interrupt
SIGQUIT	3	Core	Quit
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace/Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status
SIGPWR	19	Ignore	Power Fail/Restart
SIGWINCH	20	Ignore	Window Size Change
SIGURG	21	Ignore	Urgent Socket Condition
SIGPOLL	22	Ignore	Socket I/O Possible
SIGSTOP	23	Stop	Stopped (signal)
SIGTSTP	24	Stop	Stopped (user)
SIGCONT	25	Ignore	Continued
SIGTTIN	26	Stop	Stopped (tty input)
SIGTTOU	27	Stop	Stopped (tty output)
SIGVTALRM	28	Exit	Virtual Timer Expired
SIGPROF	29	Exit	Profiling Timer Expired
SIGXCPU	30	Core	CPU time limit exceeded
SIGXFSZ	31	Core	File size limit exceeded
SIGWAITING	32	Ignore	All LWPs blocked
SIGLWP	33	Ignore	Virtual Interprocessor Interrupt for Threads Library
SIGAIO	34	Ignore	Asynchronous I/O

UNIX processes can receive 'signals' from the user, the OS, or <u>another process</u>

e.g.

SIGHUP	1	Exit	Hangup
SIGINT	2	Exit	Interrupt
SIGQUIT	3	Core	Quit
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace/Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status
SIGPWR	19	Ignore	Power Fail/Restart
SIGWINCH	20	Ignore	Window Size Change
SIGURG	21	Ignore	Urgent Socket Condition
SIGPOLL	22	Ignore	Socket I/O Possible
SIGSTOP	23	Stop	Stopped (signal)
SIGTSTP	24	Stop	Stopped (user)
SIGCONT	25	Ignore	Continued
SIGTTIN	26	Stop	Stopped (tty input)
SIGTTOU	27	Stop	Stopped (tty output)
SIGVTALRM	28	Exit	Virtual Timer Expired
SIGPROF	29	Exit	Profiling Timer Expired
SIGXCPU	30	Core	CPU time limit exceeded
SIGXFSZ	31	Core	File size limit exceeded
SIGWAITING	32	Ignore	All LWPs blocked
SIGLWP	33	Ignore	Virtual Interprocessor Interrupt for Threads Library
SIGAIO	34	Ignore	Asynchronous I/O

UNIX processes can receive 'signals' with an associated default action

SIGHUP	1	Exit	Hangup
SIGINT	2	Exit	Interrupt
SIGQUIT	3	Core	Quit
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace/Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status
SIGPWR	19	Ignore	Power Fail/Restart
SIGWINCH	20	Ignore	Window Size Change
SIGURG	21	Ignore	Urgent Socket Condition
SIGPOLL	22	Ignore	Socket I/O Possible
SIGSTOP	23	Stop	Stopped (signal)
SIGTSTP	24	Stop	Stopped (user)
SIGCONT	25	Ignore	Continued
SIGTTIN	26	Stop	Stopped (tty input)
SIGTTOU	27	Stop	Stopped (tty output)
SIGVTALRM	28	Exit	Virtual Timer Expired
SIGPROF	29	Exit	Profiling Timer Expired
SIGXCPU	30	Core	CPU time limit exceeded
SIGXFSZ	31	Core	File size limit exceeded
SIGWAITING	32	Ignore	All LWPs blocked
SIGLWP	33	Ignore	Virtual Interprocessor Interrupt for Threads Library
SIGAIO	34	Ignore	Asynchronous I/O

1. dfr@manneback (ssh)

```c
// gcc crsigvacount.c -o crsigvacount && ./crsi
gvacount

#include <stdlib.h>
#include <signal.h>
#include <stdio.h>

volatile sig_atomic_t interrupted = 0;

void catch_signal(int sig)
{
  interrupted = 1;
}



void main()
{
  int i, the_start, the_end;
  FILE * file;

  // Register signal handler
  signal(SIGINT, catch_signal);

  // Try to recover current state
  file = fopen("state", "r");
  if (file)
  {
    fscanf(file, "%d", &the_start);
    fclose(file);
  }
  else
```

```c
  else
  {
    // Otherwise bootstrap at 1
    the_start = 1;
  }

  the_end = 10;

  for (i=the_start; i<=the_end && !interrupted;
       i++)
  {
    // Heavy computations that
    // might be interrupted
    printf("%d\n", i);
    sleep(1);
  }

  // Iterations are over or
  // have been interrupted
  // Anyway save the state.
  file = fopen("state", "w");
  fprintf(file, "%d", i);
  fclose(file);
}
```

C signal recipe

```
crsigvacount.c                15,0-1        Top crsigvacount.c                61,0-1        Bot
"crsigvacount.c" 61L, 890C written
```

3 Software to checkpoint other software

Computer Languages & Systems Software

COMPUTER LANGUAGES & SYSTEMS SOFTWARE

Research
- Pagoda
- GASNet
- UPC++
- Berkeley UPC
- Checkpoint/Restart
  - Downloads
  - Publications
  - Flyer
- DEGAS
- FastOS
- Corvette
- ASIM

Berkeley Lab Checkpoint/Restart (BLCR) for LINUX

Future Technologies Group researchers are developing a hybrid kernel/user implementation of checkpoint/restart. Their goal is to provide a robust, production quality implementation that checkpoints a wide range of applications, without requiring changes to be made to application code. This work focuses on checkpointing parallel applications that communicate through MPI and on compatibility with the software suite produced by the SciDAC Scalable Software ISIC. This work is broken down into 4 main areas:

- Checkpoint/Restart for Linux (CR)
- Checkpointable MPI Libraries
- Resource Management Interface to Checkpoint/Restart
- Development of Process Management Interfaces

TABLE OF CONTENTS

Uses a kernel module
Complex install, by root
Only beta version for CentOS7

DMTCP: Distributed MultiThreaded CheckPointing

Home
Downloads
FAQ
SF project page
Browse Source
Demo
Supported Apps
Parallel Computing
Condor Integration
Manual/Documentation
Plugins and other APIs
Publications
Contact Us

About DMTCP:

DMTCP (Distributed MultiThreaded Checkpointing) transparently checkpoints a single-host or distributed computation in user-space -- with no modifications to user code or to the O/S. It works on most Linux applications, including Python, Matlab, R, GUI desktops, MPI, etc. It is robust and widely used (on Sourceforge since 2007).

Among the applications supported by DMTCP are MPI (various implementations), OpenMP, MATLAB, Python, Perl, R, and many programming languages and shell scripting languages. With the use of TightVNC, it can also checkpoint and restart X-Window applications. The OpenGL library for 3D graphics is supported through a special plugin. It also has strong support for HPC (High Performance Computing) environments, including MPI, SLURM, InfiniBand, and other components. See QUICK-START.md for further details.

DMTCP supports the commonly used OFED API for InfiniBand, as well as its integration with various implementations of MPI, and resource managers (e.g., SLURM). See contrib/infiniband/README for more details.

News | See Also | Authors | Acknowledgment

Fully user space
Can be installed by
regular users

Main page
Recent changes
Random page
Help

Tools
What links here
Related changes
Special pages
Printable version
Permanent link
Page information

News
Google+
Twitter
YouTube

Welcome to CRIU, a project to implement checkpoint/restore functionality for Linux.

Checkpoint/Restore In Userspace, or CRIU (pronounced kree-oo, IPA: /krɪʊ/, Russian: криу), is a software tool for Linux operating system. Using this tool, you can freeze a running application (or part of it) and checkpoint it as a collection of files on disk. You can then use the files to restore the application and run it exactly as it was during the time of freeze. With this feature, application live migration, snapshots, remote debugging, and many other things are possible.

CRIU started as a project of Virtuozzo and grew with tremendous help from the community. It is currently used by (integrated into) OpenVZ, LXC/LXD, Docker, and other software, and CRIU packages is included into many Linux distributions.

Download

Tarball:	criu-3.4.tar.bz2
Version:	3.4 "Cobalt Swan"
Released:	21 Aug 2017
GIT tag:	v3.4

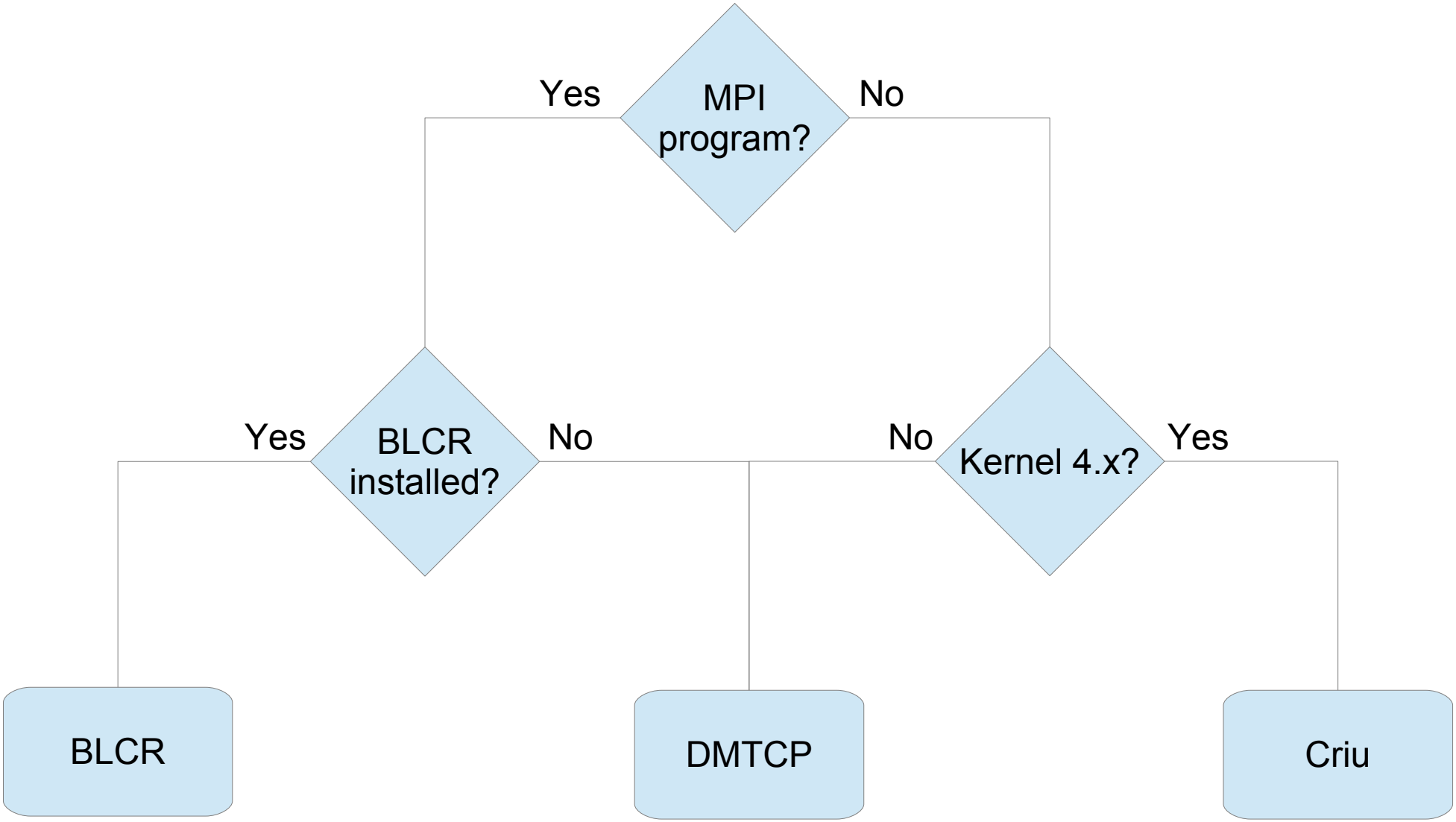
Installation • Usage

Releases • Release schedule

Uses a kernel functionality
Easy install by root
Requires kernel version 4.x

	CRIU	DMTCP	BLCR
Integration with Slurm	NO	NO* planned	YES
Requires application modification	NO	NO	Recompile app
MPI applications	NO	YES	YES
Can checkpoint running application without preloading	YES	NO	YES* library must be loaded
Overhead besides checkpoint	NONE	Init: sec. CPU: 1-2%	CPU:1-2%
Can checkpoint containers (Docker & LXD)?	YES* we have only tested Docker, not LXD	NO	NO
Infiniband support	N/A	YES	NO* we haven't tried, comes from doc.

https://slurm.schedmd.com/SLUG16/ciemat-cr.pdf


MPI program?
Yes
No
BLCR installed?
Yes
No
Kernel 4.x?
No
Yes
BLCR
DMTCP
Criu

4 Checkpointing and Slurm

Slurm integration:
scontrol checkpoint create|restart

```
checkpoint CKPT_OP ID
       Perform a checkpoint activity on the job step(s) with the specified identification.  ID can be used to identify a  spe-
       cific   job   (e.g.  "<job_id>",  which  applies  to  all  of  its  existing  steps)  or  a  specific  job  step  (e.g.
       "<job_id>.<step_id>").  Acceptable values for CKPT_OP include:

       able        Test if presently not disabled, report start time if checkpoint in progress

       create      Create a checkpoint and continue the job or job step

       disable     Disable future checkpoints

       enable      Enable future checkpoints

       error       Report the result for the last checkpoint request, error code and message

       restart     Restart execution of the previously checkpointed job or job step

       requeue     Create a checkpoint and requeue the batch job, combines vacate and restart operations

       vacate      Create a checkpoint and terminate the job or job step
Acceptable values for CKPT_OP include:

       MaxWait=<seconds>   Maximum time for checkpoint to be written.  Default value is 10 seconds.   Valid  with  create  and
                           vacate options only.

       ImageDir=<directory_name>
                           Location of checkpoint file.  Valid with create, vacate and restart options only.  This value takes
                           precedent over any --checkpoint-dir value specified at job submission time.

       StickToNodes        If set, resume job on the same nodes are previously used.  Valid with the restart option only.
```

scancel is used to send signals to jobs

```
1. dfr@manneback (ssh)
SCANCEL(1)                    Slurm components                    SCANCEL(1)

NAME
       scancel  -  Used to signal jobs or job steps that are under the control
       of Slurm.

SYNOPSIS
       scancel          [OPTIONS...]          [job_id[_array_id][.step_id]]
       [job_id[_array_id][.step_id]...]

DESCRIPTION
       scancel  is used to signal or cancel jobs, job arrays or job steps.  An
       arbitrary number of jobs or job steps may be signaled using job  speci-
       fication  filters  or a space separated list of specific job and/or job
       step IDs.  If the job ID of a job array is specified with an  array  ID
       value  then  only that job array element will be cancelled.  If the job
       ID of a job array is specified without an array ID value then  all  job
       array  elements  will be cancelled.  A job or job step can only be sig-
       naled by the owner of that job or user root.  If an attempt is made  by
       an unauthorized user to signal a job or job step, an error message will
       be printed and the job will not be signaled.

OPTIONS
:
```

--signal to have Slurm send signals automatically before the end of the allocation

1. dfr@manneback (ssh)

```
        This option may result the allocation being granted sooner  than
        if  the  --share option was not set and allow higher system uti-
        lization, but application performance will likely suffer due  to
        competition for resources within a node.

    --signal=<sig_num>[@<sig_time>]
        When  a  job is within sig_time seconds of its end time, send it
        the signal sig_num.  Due to the resolution of event handling  by
        SLURM,  the  signal  may  be  sent up to 60 seconds earlier than
        specified.  sig_num may either be a signal number or name  (e.g.
        "10"  or "USR1").  sig_time must have integer value between zero
        and 65535.  By default, no signal is sent before the  job's  end
        time.   If  a  sig_num  is  specified  without any sig_time, the
        default time will be 60 seconds.

    --sockets-per-node=<sockets>
        Restrict node selection to nodes with  at  least  the  specified
        number  of  sockets.  See additional information under -B option
        above when task/affinity plugin is enabled.

    --switches=<count>[@<max-time>]
        When a tree topology is used, this defines the maximum count  of
        switches desired for the job allocation and optionally the maxi-
/sbatch man page
```

Example: send SIGINT 60 seconds before job is killed (so, here, after 2 minutes)

```
#!/bin/bash

#SBATCH --job-name=test

#SBATCH --output=res

#SBATCH --time=0-00:03:00
#SBATCH --signal=INT@60
#SBATCH --mem-per-cpu=500

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
```

scontrol requeue

1. dfr@leleve (ssh)

```
#!/bin/bash

#SBATCH --job-name=test

#SBATCH --output=res
#SBATCH --open-mode=append

#SBATCH --time=0-00:03:00
#SBATCH --signal=INT@60
#SBATCH --mem-per-cpu=500

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1

date
echo "restarted ${SLURM_RESTART_COUNT-0} time(s)"

./crsigvalibrccount || scontrol requeue $SLURM_JOB_ID


~
~
~
~
~
~
"submit_signal_wrequeue.sh" 21L, 341C written                    1,1          All
```

Note the --open-mode=append

1. dfr@leleve (ssh)

```
#!/bin/bash

#SBATCH --job-name=test

#SBATCH --output=res
#SBATCH --open-mode=append

#SBATCH --time=0-00:03:00
#SBATCH --signal=INT@60
#SBATCH --mem-per-cpu=500

#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1

date
echo "restarted ${SLURM_RESTART_COUNT-0} time(s)"

./crsigvalibrccount || scontrol requeue $SLURM_JOB_ID



~
~
~
~
~
~
"submit_signal_wrequeue.sh" 21L, 341C written                    1,1          All
```

Or chain the jobs...

slurm
workload manager
Version 2.6

About

- Overview
- Meetings
- What's New
- Publications
- Testimonials
- SLURM Team

Using

- Tutorials
- Documentation
- FAQ
- Getting Help
- Mailing Lists

Installing

- Download
- Installation Guide
- Platforms

SchedMD

-d, --dependency=<*dependency_list*>
Defer the start of this job until the specified dependencies have been satisfied completed. <*dependency_list*> is of the form <*type:job_id[:job_id] [,type:job_id[:job_id]]*>. Many jobs can share the same dependency and these jobs may even belong to different users. The value may be changed after job submission using the scontrol command.

after:job_id[:jobid...]
This job can begin execution after the specified jobs have begun execution.

afterany:job_id[:jobid...]
This job can begin execution after the specified jobs have terminated.

afternotok:job_id[:jobid...]
This job can begin execution after the specified jobs have terminated in some failed state (non-zero exit code, node failure, timed out, etc).

afterok:job_id[:jobid...]
This job can begin execution after the specified jobs have successfully executed (ran to completion with an exit code of zero).

expand:job_id
Resources allocated to this job should be used to expand the specified job. The job to expand must share the same QOS (Quality of Service) and partition. Gang scheduling of resources in the partition is also not supported.

singleton
This job can begin execution after any previously launched jobs sharing the same job name and user have terminated.

-D, --workdir=<*directory*>
Set the working directory of the batch script to *directory* before it is executed.

Using a signal-based watchdog to re-queue the job just before it is killed

1. dfr@leleve (ssh)

```
#!/bin/bash

trap "echo TRAPPED; date; scontrol requeue $1; exit" SIGUSR1
echo Trap set. Watching...
while :
do
sleep 30
# echo Watchdog Alive
done


# in the submission script:
# srun --overcommit -n1 watchdog.sh
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"watchdog.sh" 13L, 214C                                  13,1          All
```

5 Example

DMTCP: Distributed MultiThreaded CheckPointing

About DMTCP:

DMTCP (Distributed MultiThreaded Checkpointing) is a tool to transparently checkpoint the state of multiple simultaneous applications, including multi-threaded and distributed applications. It operates directly on the user binary executable, without any Linux kernel modules or other kernel modifications.

Among the applications supported by DMTCP are Open MPI, MATLAB, Python, Perl, and many programming languages and shell scripting languages. Starting with release 1.2.0, DMTCP also supports GNU screen sessions, including vim/cscope and emacs. With the use of TightVNC, it can also checkpoint and restart X Window applications, as long as they do not use extensions (e.g.: no OpenGL, no video). See the QUICK-START file for further details.

DMTCP supports InfiniBand internally as of Aug., 2013, and will soon be released.

DMTCP is also the basis for URDB, the Universal Reversible Debugger. URDB was an experimental project for reversibility for four debuggers: gdb, MATLAB, python (pdb), and perl (perl -d). It is now obsolete, and work is continuing on a newer internal project, which will be released as open source in the future.

News | See Also | Authors | Acknowledgement

Announcement!

We are currently looking for well qualified applicants who are interested in joining a Ph.D. program in order to do research on checkpointing and reversible debugging. Interested applicants should write to Gene Cooperman (gene@ccs.neu.edu) at Northeastern University.

Advertised Features

- Distributed Multi-Threaded CheckPointing
- Works with Linux Kernel 2.6.9 and later
- Supports sequential and multi-threaded computations across single/multiple hosts
- Entirely in user space (no kernel modules or root privilege)
- Transparent (no recompiling, no re-linking)
- Written at Northeastern U. and MIT and under active development for 5+ years
- LGPL'd and freely available
- No remote I/O
- Supports threads, mutexes/semaphoes, forks, shared memory, exec, and many more

From their FAQ:

“**What types of programs can DMTCP checkpoint?**
It checkpoints most binary programs on most Linux distributions. Some examples on which users have verified that DMTCP works are: Matlab, R, Java, Python, Perl, Ruby, PHP, Ocaml, GCL (GNU Common Lisp), emacs, vi/cscope, Open MPI, MPICH-2, OpenMP, and Cilk. See Supported Applications for further details. Our goal is to support DMTCP for all vanilla programs. If DMTCP does not work correctly on your program, **then this is a bug in DMTCP**. We would be appreciative if you can then file a bug report with DMTCP.”

Imagine a non-checkpointable program

1. dfr@manneback (ssh)

```
// gcc count.c -o count && ./count
#include <stdio.h>
void main()
{
  int i, the_start, the_end;

  the_start = 1;
  the_end = 10;

  for (i=the_start; i<=the_end; i++)
  {
    printf("%d\n", i);
    sleep(1);
  }
}
~
~
~
~
~
~
~
~
"count.c" 15L, 219C                    1,1          All
```

Run with dmtcp_launch (runs monitoring daemon if necessary)

1. dfr@leleve (ssh)

```
dfr@leleve:~/Checkpointing $ dmtcp_launch ./count & sleep 4 ; dmtcp_command --quiet
--checkpoint ; sleep 1 ; dmtcp_command --quiet --quit
[1] 2976
dmtcp_launch (DMTCP + MTCP) version 2.0
Copyright (C) 2006-2013  Jason Ansel, Michael Rieker, Kapil Arya, and
                                                        Gene Cooperman
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions; see COPYING file for details.
(Use flag "-q" to hide this message.)

dmtcp_coordinator starting...
    Host: leleve.cism.ucl.ac.be (0.0.0.0)
    Port: 7779
    Checkpoint Interval: disabled (checkpoint manually instead)
    Exit on last client: 1
Backgrounding...
1
2
3
4
5
[1]+  Done                    dmtcp_launch ./count
dfr@leleve:~/Checkpointing $ ls -rtl|tail -1
-rwxrw-r--  1 dfr dfr    5167 Oct 15 11:51 dmtcp_restart_script_1dcda56f5a2723b6-40000-
525d1005.sh
dfr@leleve:~/Checkpointing $
```

Restart with dmtcp_restart_script.sh

```
1. dfr@leleve (ssh)
5
[1]+  Done                    dmtcp_launch ./count
dfr@leleve:~/Checkpointing $ ls -rtl|tail -1
-rwxrw-r--  1 dfr dfr    5167 Oct 15 11:52 dmtcp_restart_script_1dcda56f5a2723b6-40000-
525d1043.sh
dfr@leleve:~/Checkpointing $ ./dmtcp_restart_script.sh
dmtcp_restart (DMTCP + MTCP) version 2.0
Copyright (C) 2006-2013  Jason Ansel, Michael Rieker, Kapil Arya, and
                                                      Gene Cooperman
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions; see COPYING file for details.
(Use flag "-q" to hide this message.)

dmtcp_coordinator starting...
    Host: leleve.cism.ucl.ac.be (0.0.0.0)
    Port: 7779
    Checkpoint Interval: disabled (checkpoint manually instead)
    Exit on last client: 1
Backgrounding...
5
6
7
8
9
10
^C
dfr@leleve:~/Checkpointing $
```

Launch the coordinator and the program with automatic checkpointing every 30 seconds

1. dfr@leleve (ssh)

```
#!/bin/sh
# Sample SLURM batch script to run a program
# Be sure to modify the XXXX to the actual number of tasks for --ntasks.
# Be sure to also modify the dmtcp_launch line for your actual job.

#SBATCH --ntasks=1
#SBATCH --time=2
#SBATCH --output=res

# Report actual hostname to user.
hostname

# If you install DMTCP in your user directory (not cluster-wide) you need to
# extend the PATH variable:
export PATH=./dmtcp-2.0/bin:$PATH

# Start dmtcp_coordinator (Fix if debugging or using coordinator on front end.)
srun --overcommit --ntasks=1 dmtcp_coordinator &

# DMTCP coordinator needs to be started on localhost. Or put the other host
#   in the '-h' option.
# The flag '--interval 3600' creates a checkpoint every hour (3600 seconds).
dmtcp_launch --rm --interval 30 -h `hostname` ./count
~
~
~
~
:noh                                                    1,1          All
```

Launch coordinator and restart program

1. dfr@leleve (ssh)

```
#!/bin/sh
# Sample SLURM batch script for restart
# You'll also need to customize the SBATCH lines below.
# The script, ./dmtcp_restart_script.sh, will have been created for you
#   by a checkpoint during the initial run.

#SBATCH --ntasks=1
#SBATCH --output=res
#SBATCH --open-mode=append

# Report actual hostname to user.
hostname

# If you install DMTCP in your user directory (not cluster-wide), you'll
# need to extend PATH variable:
export PATH=./dmtcp-2.0/bin:$PATH

# Start dmtcp_coordinator (Fix if debugging or using coordinator on front end.)
srun --overcommit dmtcp_coordinator &
export DMTCP_HOST=`hostname`

# The flag '--interval 3600' creates a checkpoint every hour (3600 seconds).
./dmtcp_restart_script.sh --interval 30
~
~
~
~
"submit.dmtcp.restart.sh" 23L, 742C                    20,1          All
```

```
1. dfr@leleve (ssh)
leleve01.cism.ucl.ac.be
dmtcp_launch (DMTCP + MTCP) version 2.0
[...]
dmtcp_coordinator starting...
Backgrounding...
1
srun: error: leleve01: task 0: Exited with exit code 99
2
3
[...]
16
17
slurmd[leleve01]: error: *** JOB 93 CANCELLED AT 2013-10-15T15:28:04 DUE TO TIME LIMIT
***
leleve01.cism.ucl.ac.be
dmtcp_restart (DMTCP + MTCP) version 2.0
[...]
dmtcp_coordinator starting...
Backgrounding...
srun: error: leleve01: task 0: Exited with exit code 99
17
18
19
[...]
25
26
27
"res" 29 lines --3%--                                   1,1          Top
```

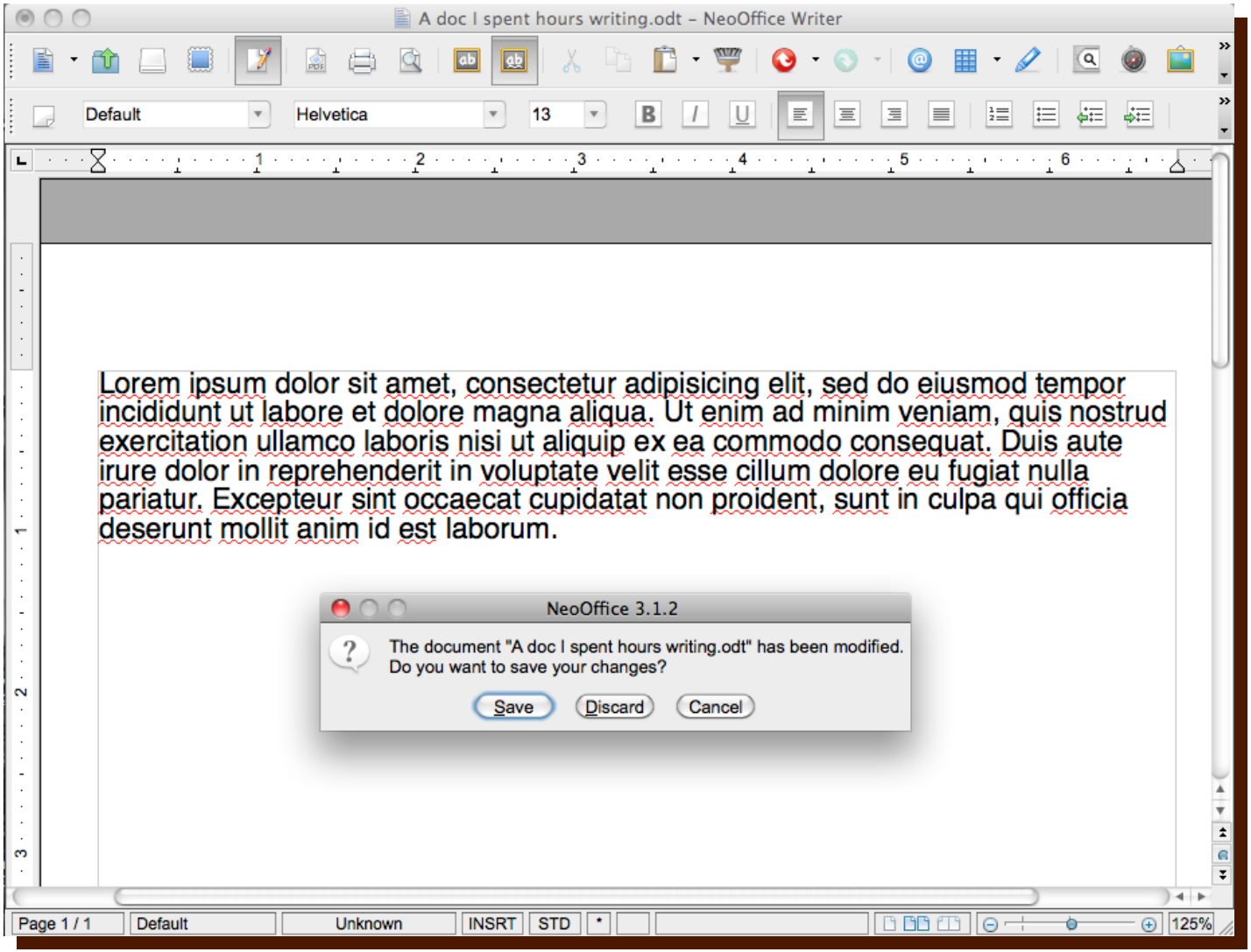
https://github.com/dmtcp/dmtcp/blob/master/plugin/batch-queue/job_examples/slurm_launch.job
https://github.com/dmtcp/dmtcp/blob/master/plugin/batch-queue/job_examples/slurm_rstr.job

damien.francois@uclouvain.be
UCL/CISM

Summary, Wrap-up and Conclusions.

October 2014
CISM/CÉCI training session

Never click 'Discard' again...



The submission script(s)

- Either one big one or two small ones
- Checkpoint periodically or --signal
- Requeue automatically
- Open-mode=append