

Plateforme technologique de Calcul Intensif et Stockage de Masse



BELGIUM

Checkpointing

Olivier Mattelaer

What is checkpointing



\$./count

\$./count

\$./count

\$./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

I

Without checkpointing:

\$./count 1 2 3^C \$./count 1 2 With checkpointing:

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

I

Without checkpointing: \$./count 2 3vC \$./count 2 3

With checkpointing:

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

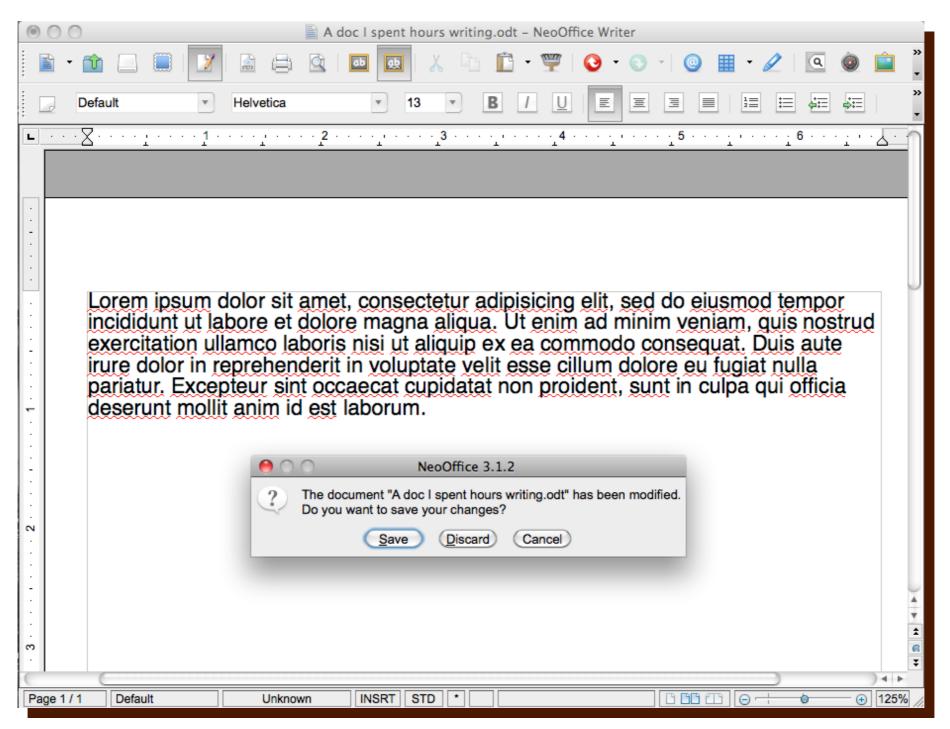
Without checkpointing: U With checkpointing: \$./countCheckpointing:/count saving' a computation so that it can be resumed later (rather than started again)

Today's agenda:

General concepts and scientific soft.
 Working with Signals
 Slurm recipes
 DMTCP

Why do we need checkpointing

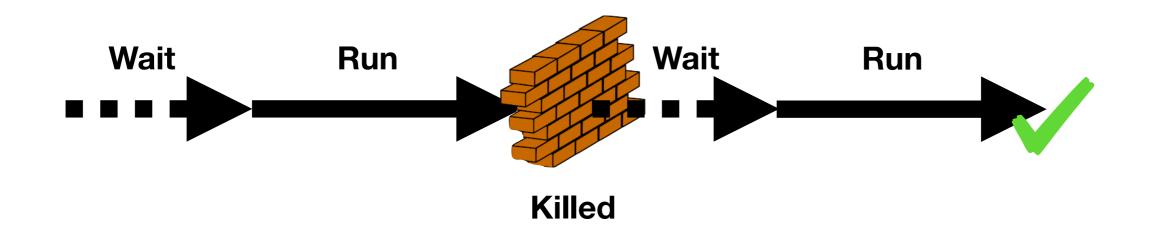
Imagine a text editor without 'checkpointing' ...

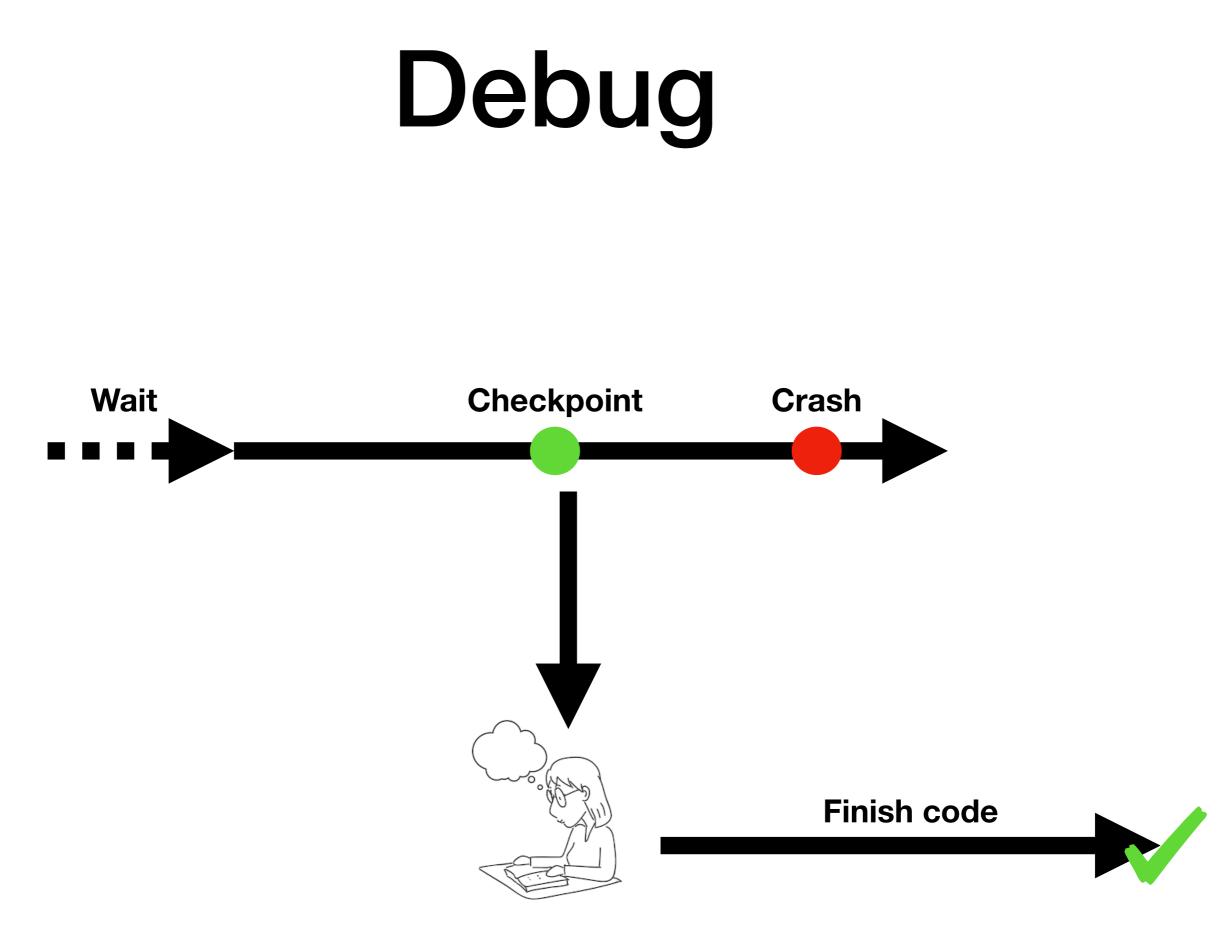


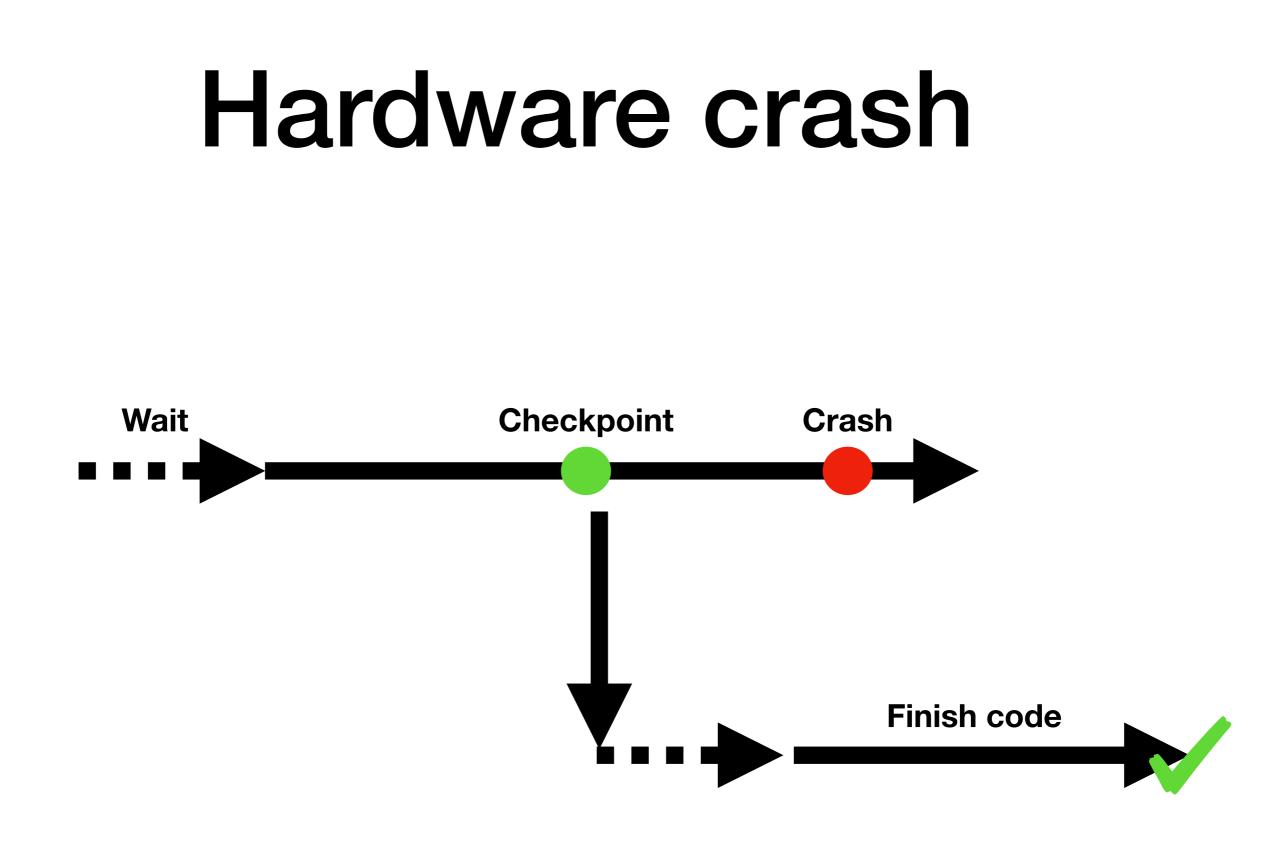
Goals of checkpointing in HPC:

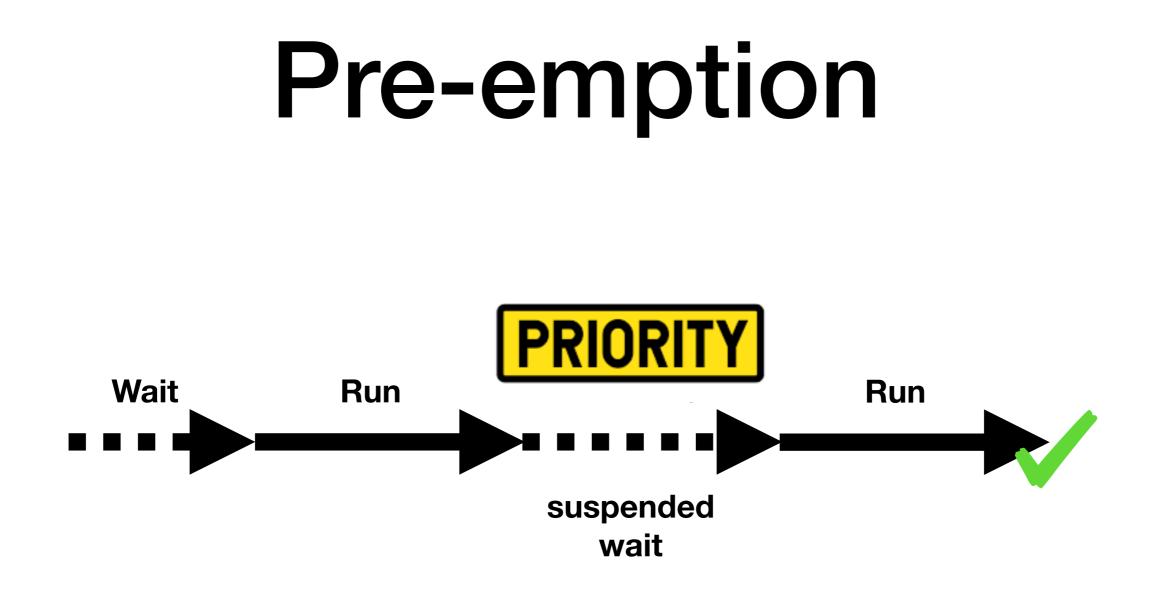
- 1. Fit in time constraints
- 2. Debugging, monitoring
- 3. Cope with hardware failures
- 4. Job preemption

Wall-Time







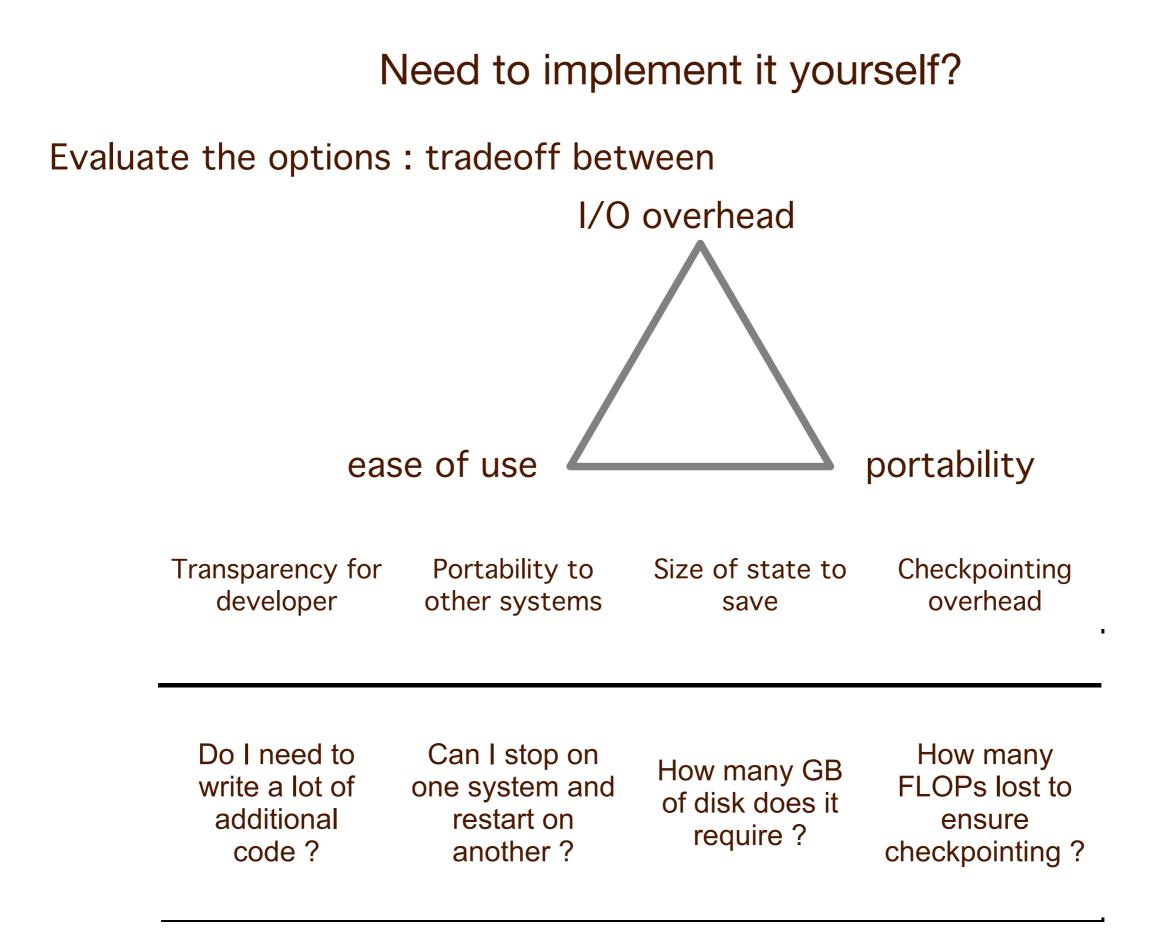


Checkpointing with scientific software Do they support checkpointing?

Working with checkpoint-restart-able software

Many scientific software have built-in checkpointing capabilities (although it might not be called that way)

Check the documentation



Demo #1

count.py Save state at each iteration

Using UNIX signals to reduce overhead : do not save the state at each iteration -- wait for the signal.

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	
	_		28	

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

^D

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	- kill -9
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	1— kill
SIGUSR1	16	Exit	User Signal 1	1
SIGUSR2	17	Exit	User Signal 2	7
SIGCHLD	18	Ignore	Child Status	7
SIGPWR	19	Ignore	Power Fail/Restart	7
SIGWINCH	20	Ignore	Window Size Change	7
SIGURG	21	Ignore	Urgent Socket Condition	7
SIGPOLL	22	Ignore	Socket I/O Possible	7
SIGSTOP	23	Stop	Stopped (signal)	7
SIGTSTP	24	Stop	Stopped (user)	f f a ha
SIGCONT	25	Ignore	Continued	fg, bg
SIGTTIN	26	Stop	Stopped (tty input)	1
SIGTTOU	27	Stop	Stopped (tty output)	7
SIGVTALRM	28	Exit	Virtual Timer Expired	7
SIGPROF	29	Exit	Profiling Timer Expired	7
SIGXCPU	30	Core	CPU time limit exceeded	7
SIGXFSZ	31	Core	File size limit exceeded	1
SIGWAITING	32	Ignore	All LWPs blocked	1
SIGLWP	33	Ignore	Virtual Interprocessor Interrupt for Threads Library	1
SIGAIO	34	Ignore	Asynchronous I/O 29	7

^Z –

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

e.g.

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

SIGHUP	1	Exit	Ha	ngup
SIGINT	2	Exit	Int	errupt
SIGQUIT	3	Core	Qu	it
SIGILL	4	Core	111	gal Instruction
SIGTRAP	5	Core	Tr	ce/Breakpoint Trap
SIGABRT	6	Core	Ał	ort
SIGEMT	7	Core	Er	ulation Trap
SIGFPE	8	Core	Ar	thmetic Exception
SIGKILL	9	Exit	Ki	led
SIGBUS	10	Core	Bι	s Error
SIGSEGV	11	Core	Se	mentation Fault
SIGSYS	12	Core	Ba	d System Call
SIGPIPE	13	Exit	Br	oken Pipe
SIGALRM	14	Exit	Al	ırm Clock
SIGTERM	15	Exit	Te	minated
SIGUSR1	16	Exit	Us	er Signal 1
SIGUSR2	17	Exit	Us	er Signal 2
SIGCHLD	18	Ignore	Cł	ild Status
SIGPWR	19	Ignore	Po	wer Fail/Restart
SIGWINCH	20	Ignore	W	ndow Size Change
SIGURG	21	Ignore	Ur	ent Socket Condition
SIGPOLL	22	Ignore	So	cket I/O Possible
SIGSTOP	23	Stop	St	pped (signal)
SIGTSTP	24	Stop	St	pped (user)
SIGCONT	25	Ignore	Co	ntinued
SIGTTIN	26	Stop	St	pped (tty input)
SIGTTOU	27	Stop	St	pped (tty output)
SIGVTALRM	28	Exit	Vi	tual Timer Expired
SIGPROF	29	Exit	Pr	filing Timer Expired
SIGXCPU	30	Core	CI	U time limit exceeded
SIGXFSZ	31	Core	Fil	e size limit exceeded
SIGWAITING	32	Ignore	Al	LWPs blocked
SIGLWP	33	Ignore	Vi	tual Interprocessor Interrupt for Threads Library
SIGAIO	34	Ignore	As	/nchronous I/O
				- 31

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

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	12	Exit		
			Broken Pipe Alarm Clock	
SIGALRM	14	Exit	Terminated	
SIGUSR1	16	Exit	User Signal 1	
SIGUSR2	17	Exit	User Signal 2	
SIGCILD	10	ignore		
SIGPWR	19	Ignore	Power Fail/Restart	
SIGWINCH	20	Ignore	Window Size Change	
			Urgent Socket Condition	
SIGURG	21	Ignore	-	
SIGURG SIGPOLL	22	Ignore	Socket I/O Possible	
SIGURG SIGPOLL SIGSTOP			-	
SIGURG SIGPOLL	22 23 24	Ignore	Socket I/O Possible	
SIGURG SIGPOLL SIGSTOP	22 23	Ignore Stop	Socket I/O Possible Stopped (signal)	
SIGURG SIGPOLL SIGSTOP SIGTSTP	22 23 24	Ignore Stop Stop	Socket I/O Possible Stopped (signal) Stopped (user)	
SIGURG SIGPOLL SIGSTOP SIGTSTP SIGCONT	22 23 24 25	Ignore Stop Stop Ignore	Socket I/O Possible Stopped (signal) Stopped (user) Continued	
SIGURG SIGPOLL SIGSTOP SIGTSTP SIGCONT SIGTTIN	22 23 24 25 26	Ignore Stop Stop Ignore Stop	Socket I/O Possible Stopped (signal) Stopped (user) Continued Stopped (tty input)	
SIGURG SIGPOLL SIGSTOP SIGTSTP SIGCONT SIGTTIN SIGTTOU	22 23 24 25 26 27	Ignore Stop Stop Ignore Stop Stop	Socket I/O Possible Stopped (signal) Stopped (user) Continued Stopped (tty input) Stopped (tty output)	
SIGURG SIGPOLL SIGSTOP SIGTSTP SIGCONT SIGTTIN SIGTTOU SIGVTALRM	22 23 24 25 26 27 28	Ignore Stop Stop Ignore Stop Stop Exit	Socket I/O Possible Stopped (signal) Stopped (user) Continued Stopped (tty input) Stopped (tty output) Virtual Timer Expired	
SIGURG SIGPOLL SIGSTOP SIGTSTP SIGCONT SIGTTIN SIGTTOU SIGVTALRM SIGPROF	22 23 24 25 26 27 28 29	Ignore Stop Stop Ignore Stop Stop Exit Exit	Socket I/O Possible Stopped (signal) Stopped (user) Continued Stopped (tty input) Stopped (tty output) Virtual Timer Expired Profiling Timer Expired	
SIGURG SIGPOLL SIGSTOP SIGTSTP SIGCONT SIGTTIN SIGTTOU SIGTTOU SIGVTALRM SIGPROF SIGXCPU	22 23 24 25 26 27 28 29 30	Ignore Stop Stop Ignore Stop Stop Exit Exit Core	Socket I/O Possible Stopped (signal) Stopped (user) Continued Stopped (tty input) Stopped (tty output) Virtual Timer Expired Profiling Timer Expired CPU time limit exceeded	
SIGURG SIGPOLL SIGSTOP SIGTSTP SIGCONT SIGTTIN SIGTTOU SIGVTALRM SIGPROF SIGXCPU SIGXFSZ	22 23 24 25 26 27 28 29 30 31	Ignore Stop Stop Ignore Stop Stop Exit Exit Core Core	Socket I/O Possible Stopped (signal) Stopped (user) Continued Stopped (tty input) Stopped (tty output) Virtual Timer Expired Profiling Timer Expired CPU time limit exceeded File size limit exceeded	

Demo #2

count-signal.py Catch control-C to save state

Slurm signaling abilities to manage checkpoint-able software in Slurm scripts on the clusters.

scancel is used to send signals to jobs

00	1. dfr@manneback (ssh)	(
SCANCEL	L(1) Slurm components	SCANCEL(1)
IAME		
	scancel - Used to signal jobs or job steps that are under of Slurm.	r the control
YNOPSI	IS	
	<pre>scancel [OPTIONS] [job_id[_array_i [job_id[_array_id][.step_id]]</pre>	.d][.step_id]]
ESCRIF	PTION	
	scancel is used to signal or cancel jobs, job arrays or j arbitrary number of jobs or job steps may be signaled usin fication filters or a space separated list of specific j step IDs. If the job ID of a job array is specified with value then only that job array element will be cancelled ID of a job array is specified without an array ID value to array elements will be cancelled. A job or job step can naled by the owner of that job or user root. If an attemp an unauthorized user to signal a job or job step, an error be printed and the job will not be signaled.	ng job speci- job and/or job an array ID d. If the job then all job n only be sig- ot is made by
PTIONS		

scancel -s SIGINT JOBID

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

x root@lm3-m001:~ (ssh)

AllowSpecResourcesUsage is enabled, the job will be allowed to override CoreSpecCount and use the specialized resources on nodes it is allocated. This option can not be used with the --thread-spec option.

--signal=[B:]<sig_num>[@<sig_time>]

When a job is within <u>sig_time</u> seconds of its end time, send it the signal <u>sig_num</u>. Due to the resolution of event handling by Slurm, the signal may be sent up to 60 seconds earlier than specified. <u>sig_num</u> may either be a signal number or name (e.g. "10" or "USR1"). <u>sig_time</u> must have an integer value between 0 and 65535. By default, no signal is sent before the job's end time. If a <u>sig_num</u> is specified without any <u>sig_time</u>, the default time will be 60 seconds. Use the "B:" option to signal only the batch shell, none of the other processes will be signaled. By default all job steps will be signaled, but not the batch shell itself.

--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.

--spread-job

Spread the job allocation over as many nodes as possible and attempt to evenly distribute tasks across the allocated nodes. This option disables the topology/tree plugin.

--signal=B:SIGINT send signal to the bash script --signal=SIGINT send signal to the srun command

Note the --open-mode=append

\times root@lm3-m001:~ (ssh)
File Edit Options Buffers Tools Sh-Script Help
#!/bin/bash
#SBATCHjob-name=test
#SBATCHoutput=test.signal
#SBATCHopen-mode=append
#SBATCHtime=0-00:03:00
#SBATCHsignal=SIGINT@60
#SBATCHntasks=1
#SBATCHpartition=debug
date
echo "restarted \${SLURM_RESTART_COUNT-0}" module load Python/2.7.14-foss-2017b
pythonversion
srunovercommit -n1 python ./count-signal.py
stant over commete the pychon ./ counce stighter.py

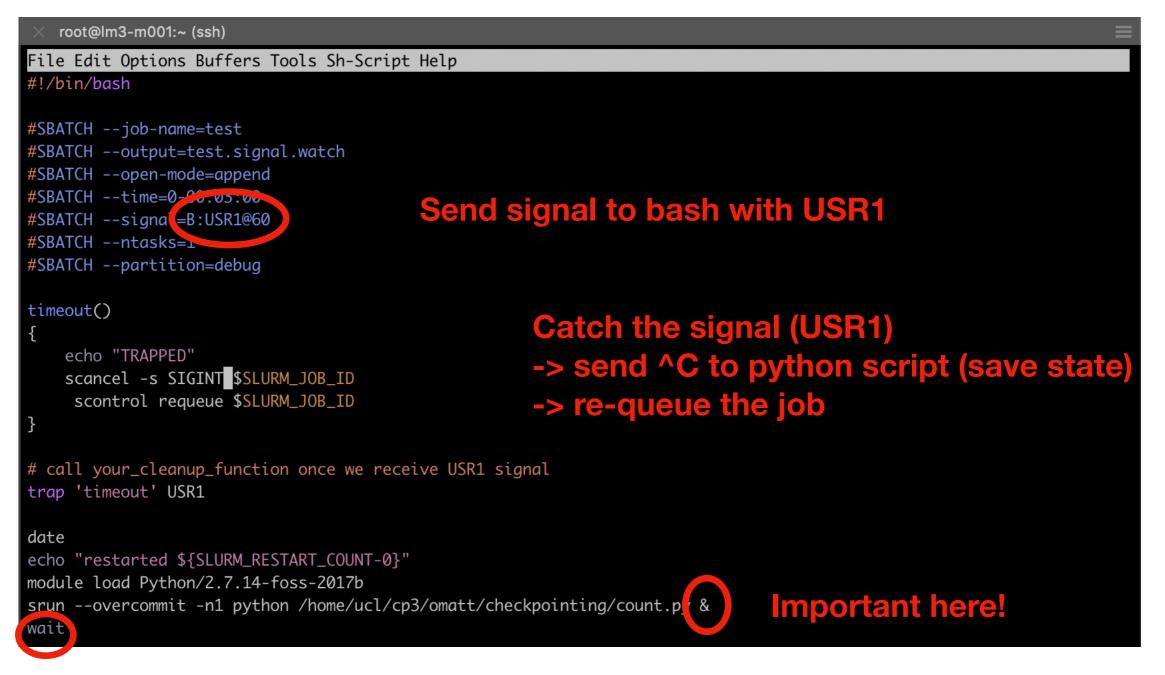
Note that we need the srun here

Demo #3

submit-signal.sh python: Catch control-C to save state Slurm send control-C between 1 and 2 minutes

submit-signal2.sh python: Catch control-C to save state Slurm send control-C between 1 and 2 minutes Automatic re-queuing

Adding requeuing automatically



Demo #4

slurm-signal-3.sh

Slurm send USR1 between 1 and 2 minutes Bash catch the message send Ctrl-c to python python: Catch control-C to save state Automatic resubmission

Making non restartable software restartable with DMTCP

DMTCP: Distributed MultiThreaded CheckPointing

About DMTCP:

DMTCP (Distributed MultiThreaded Checkpointing) transparently checkpoints a single-host or distributed computation in user-space -- with no modifications to user code r to the O/S. It works on most Linux applications, including Python, Matlab, R, GUI desktops, MPI, etc. It is rebust 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 restort 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, an other components. See <u>QUICK-START.md</u> 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 <u>contrib/infiniband/README</u> for more details.

News | See Also | Authors | Acknowledgment

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 with applications to HPC, supercomputing, cloud computing, security, and other areas. Interested applicants should write to Gene Cooperman (gene@ccs.neu.edu) at Northeastern University.

News

[2019-08-14]: Upcoming releases:

1. A totally revised DMTCP module for support of MPI is planned (based on MANA; MPI-Agnostic, Network-Agnostic; see DMTCP Publications).

2. Longer-term, a DMTCP version 3.0 is being prepared with new and better features (e.g., user-programmable barriers within a plugin). If you would like to try it now, see Downloads.

[2019-08-14]: DMTCP 2.6.0 released!

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 $5+\ensuremath{\mathsf{years}}$
- 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

```
00
                                   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++)</pre>
  Ł
    printf("%d\n", i);
    sleep(1);
"count.c" 15L, 219C
                                                                  1.1
                                                                                  A11
```

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 [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)
[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
AC.
dfr@leleve:~/Checkpointing $
```

Apply it for Slurm

Lemaitre3 specific!

1. Start DMTCP coordinator start_coordinator -i 10 # -i 120 ... <put dmtcp coordinator options here> 2. Launch application 2.1. If you use mpiexec/mpirun to launch an application, use the following command line: \$ dmtcp_launch --rm mpiexec <mpi-options> ./<app-binary> <app-options> 2.2. If you use PMI1 to launch an application, use the following command line: \$ srun dmtcp_launch --rm ./<app-binary> <app-options> # Note: PMI2 is not supported yet. 2.3. If you use the Stampede supercomputer at Texas Advanced Computing Center (TACC), use ibrun command to launch the application (--rm is not required): \$ ibrun dmtcp_launch ./<app-binary> <app-options> srun dmtcp_launch -allow-file-overwrite --rm python -u count-orig.py 10<&- 11>&-Slurm aware 47

start coordinator

Normal job with decorator

Resubmit

Launch application#
######################################

start_coordinator -i 10 -i 120 <put coordinator="" dmtcp="" here="" options=""></put>

2. Restart application
<pre>/bin/bash ./dmtcp_restart_script.sh -h \$DMTCP_COORD_HOST -p \$DMTCP_COORD_PORT</pre>

<pre># If you use the Stampede supercomputer at Texas Advanced Computing Center # (TACC) add the heatfile entires</pre>
<pre># (TACC), add thehostfile option: # /bin/bash ./dmtcp_restart_script.sh -h \$DMTCP_COORD_HOST -p \$DMTCP_COORD_PORT\</pre>
#hostfile \$HOSTFILE

start coordinator

Script created by previous run

Let's combine everything

Use DMTCP with periodic check add an additional checkpoint before wall time Auto resubmit

# <mark>!/bin/bash</mark>	
<pre># Put your SLURM options here</pre>	
#SBATCHpartition=debug	# change to proper partition name or remove
#SBATCHtime=00:00:30	<pre># put proper time of reservation here</pre>
#SBATCHnodes=1	# number of nodes
#SBATCHntasks-per-node=1	# processes per node
<pre>#SBATCHjob-name="dmtcp_job"</pre>	# change to your job name
<pre>#SBATCHoutput=slurm.dmtcp</pre>	# change to proper file name or remove for defaults
<pre>#SBATCHsignal=B:USR1@60</pre>	
#SBATCH open-mode=append	
#######################################	#######################################
<pre># 1. Start DMTCP coordinator</pre>	
#######################################	#######################################
<pre>start_coordinator -i 10 # -i 120 <put d<="" pre=""></put></pre>	dmtcp coordinator options here>
#######################################	#######################################
# 2. Launch application	
	<i>#####################################</i>
<pre>echo "requeue #\${SLURM_RESTART_COUNT}"</pre>	
if TT a dater vestant seriet sh 88 "\${{}	
<pre>if [[-e dmtcp_restart_script.sh && "\${SLUF</pre>	<pre>\$DMTCP_COORD_HOST -p \$DMTCP_COORD_PORT &</pre>
else	SDMTCP_COOKD_HOST -p SDMTCP_COOKD_FORT &
	riterm python -u count-orig.py 10<&- 11>&- &
fi	The second of th
#######################################	#######################################
# 3. setup requeue for the wall time	
<pre># Note the #SBATCHsignal=B:USR1@60 which</pre>	h is needed
	#######################################
<pre>timeout(){</pre>	
echo "doing checkpoint"	
dmtcp_commandcheckpoint	
sleep 2	
echo "doing checkpoint; done"	
dmtcp_commandquit	
sleep 2	
<pre>scontrol requeue \$SLURM_JOB_ID</pre>	
}	

```
trap 'timeout' USR1
wait
```

wait

Periodic checkpoint

#!/bin/bash	
<pre># Put your SLURM options here</pre>	
#SBATCHpartition=debug	# change to proper partition name or remove
#SBATCHtime=00:00:30	# put proper time of reservation here
#SBATCHnodes=1	# number of nodes
	# processes per node
	# change to your job name
	# change to proper file name or remove for default
#SBATCHsignal=B:USR1@60	
#SBATCHopen-mode=append	
	#######################################
# 1. Start DMTCP coordinator	
	#######################################
start_coordinator -i 10 🐔 -i 120 <put dm<="" th=""><th>tcp coordinator options here></th></put>	tcp coordinator options here>
# 2. Launch application	***************************************
# 2. Edunen appriedeton ####################################	****
echo "requeue #\${SLURM_RESTART_COUNT}"	
<pre>if [[-e dmtcp_restart_script.sh && "\${SLURM</pre>	_RESTART_COUNT}" != ""]]; then
<pre>/bin/bash ./dmtcp_restart_script.sh -h \$</pre>	DMTCP_COORD_HOST -p \$DMTCP_COORD_PORT &
else	
	term python -u count-orig.py 10<&- 11>&- &
fi	
######################################	***************************************
<pre># S. setup requeue for the wall time # Note the #SBATCHsignal=B:USR1@60 which</pre>	is needed
<pre>timeout(){</pre>	
echo "doing checkpoint"	
dmtcp_commandcheckpoint	
sleep 2	
echo "doing checkpoint; done"	
dmtcp_commandquit	
sleep 2	
<pre>scontrol requeue \$SLURM_JOB_ID</pre>	
}	
trap 'timeout' USR1	

Periodic checkpoint Checkpoint at walltime

```
/bin/bash
          Put your SLURM options here
         SBATCH --partition=debug
                                        # change to proper partition name or remove
         SBATCH --time=00:00:30
                                        # put proper time of reservation here
         SBATCH --nodes=1
                                        # number of nodes
          SBATCH --ntasks-per-node=1
                                        # processes per node
          SBATCH --job-name="dmtcp_job"
                                        # change to your job name
         #SRATCH -- out nut-slurm dmtcn
                                        # change to proper file name or remove for defaults
         #SBATCH --signal=B:USR1@60
         #SBATCH --open-mode=append
 1. Start DMTCP coordinator
    ****
start_coordinator -i 10 # -i 120 ... <put dmtcp coordinator options here>
# 2. Launch application
echo "requeue #${SLURM_RESTART_COUNT}"
if [[ -e dmtcp_restart_script.sh && "${SLURM_RESTART_COUNT}" != "" ]]; then
   /bin/bash ./dmtcp_restart_script.sh -h $DMTCP_COORD_HOST -p $DMTCP_COORD_PORT &
else
     srun dmtcp_launch --allow-file-overwrite --rm python -u count-orig.py 10<&- 11>&- &
fi
 3. setup requeue for the wall time
 Note the #SBATCH --signal=B:USR1@60 which is needed
timeout(){
ecno aoing cneckpoint"
dmtcp_command --checkpoint
sleep 2
echo "doing checkpoint; done"
dmtcp_command --quit
sleep 2
```

scontrol requeue \$SLURM_JOB_ID

trap 'timeout' USR1
wait

Periodic checkpoint Checkpoint at walltime

else

sleep 2

sleep 2

wait

dmtcp_command --checkpoint

echo "doing checkpoint; done"

scontrol requeue \$SLURM_JOB_ID

dmtcp_command --quit

trap 'timeout' USR1

fi

```
!/bin/bash
          Put your SLURM options here
         SBATCH --partition=debug
                                       # change to proper partition name or remove
         SBATCH --time=00:00:30
                                       # put proper time of reservation here
         #SBATCH --nodes=1
                                       # number of nodes
         #SBATCH --ntasks-per-node=1
                                       # processes per node
         SBATCH --job-name="dmtcp_job"
                                       # change to your job name
         #SBATCH --output=slurm.dmtcp
                                       # change to proper file name or remove for defaults
         #SBATCH --signal=B:USR1@60
          SBATCH --open-mode=append
 1. Start DMTCP coordinator
    ****
start_coordinator -i 10 # -i 120 ... <put dmtcp coordinator options here>
# 2. Launch application
echo "requeue #${SLURM_RESTART_COUNT}"
if [[ -e dmtcp_restart_script.sh && "${SLURM_RESTART_COUNT}" != "" ]]; then
   /bin/bash ./dmtcp_restart_script.sh -h $DMTCP_COORD_HOST -p $DMTCP_COORD_PORC &
     srun dmtcp_launch --allow-file-overwrite --rm python -u count-orig.py 10<&- 11>& &
 3. setup requeue for the wall time
# Note the #SBATCH --signal=B:USR1@60 which is needed
timeout(){
echo "doing checkpoint"
```

Periodic checkpoint Checkpoint at walltime

trap 'timeout' USR1

wait

Auto-resubmit

#!/bin/bash			
<pre># Put your SLURM options here</pre>			
#SBATCHpartition=debug	# change to proper partition name or remove		
#SBATCHtime=00:00:30	<pre># put proper time of reservation here</pre>		
#SBATCHnodes=1	# number of nodes		
#SBATCHntasks-per-node=1			
<pre>#SBATCHjob-name="dmtcp_job"</pre>	# change to your job name		
<pre>#SBATCHoutput=slurm.dmtcp</pre>	# change to proper file name or remove for default		
<pre>#SBATCHsignal=B:USR1@60</pre>			
#SBATCHopen-mode=append			
#######################################	#######################################		
# 1. Start DMTCP coordinator			
#######################################	#######################################		
<pre>start_coordinator -i 10 # -i 120 <put @<="" pre=""></put></pre>	dmtcp coordinator options here>		
#######################################	#######################################		
# 2. Launch application			
#######################################	#######################################		
<pre>echo "requeue #\${SLURM_RESTART_COUNT}"</pre>			
if [[-e dmtcp_restart_script.sh && "\${SLURM_RESTART_COUNT}" != ""]]; then			
<pre>/bin/bash ./dmtcp_restart_script.sh -h \$DMTCP_COORD_HOST -p \$DMTCP_COORD_PORT &</pre>			
else			
srun dmtcp_launchallow-file-overwriterm python -u count-orig.py 10<&- 11>&- &			
fi			
#######################################	#######################################		
# 3. setup requeue for the wall time			
<pre># Note the #SBATCHsignal=B:USR1@60 which is needed</pre>			
#######################################			
<pre>timeout(){</pre>			
echo "doing checkpoint"			
dmtcp_commandcheckpoint			
sleep 2			
echo "doing checkpoint; done"			
dmtcp_commandquit			
sloop 2			
<pre>scontrol requeue \$SLURM_JOB_ID</pre>			
}			

S

Periodic checkpoint Checkpoint at walltime

Auto-resubmit

Additional

```
!/bin/bash
          Put your SLURM options here
          SBATCH --partition=debug
                                        # change to proper partition name or remove
          SBATCH --time=00:00:30
                                        # put proper time of reservation here
         #SBATCH --nodes=1
                                        # number of nodes
         $BATCH --ntasks-per-node=1
                                        # processes per node
          SBATCH --job-name="dmtcp_job"
                                        # change to your job name
                                        # change to proper file name or remove for defaults
         #SBATCH --output=slurm.dmtcp
         #SRATCH -- signal -R·USR1@60
         #SBATCH --open-mode=append
 1. Start DMTCP coordinator
    *****
start_coordinator -i 10 # -i 120 ... <put dmtcp coordinator options here>
# 2. Launch application
echo "requeue #${SLURM_RESTART_COUNT}"
if [[ -e dmtcp_restart_script.sh && "${SLURM_RESTART_COUNT}" != "" ]]; then
   /bin/bash ./dmtcp_restart_script.sh -h $DMTCP_COORD_HOST -p $DMTCP_COORD_PORT &
else
     srun dmtcp_launch --allow-file-overwrite --rm python -u count-orig.py 10<&- 11>&- &
fi
 3. setup requeue for the wall time
# Note the #SBATCH --signal=B:USR1@60 which is needed
timeout(){
echo "doing checkpoint"
dmtcp_command --checkpoint
sleep 2
echo "doing checkpoint; done"
dmtcp_command --quit
sleep 2
scontrol requeue $SLURM_JOB_ID
```

trap 'timeout' USR1

wait

Periodic checkpoint Checkpoint at walltime

wait

Auto-resubmit

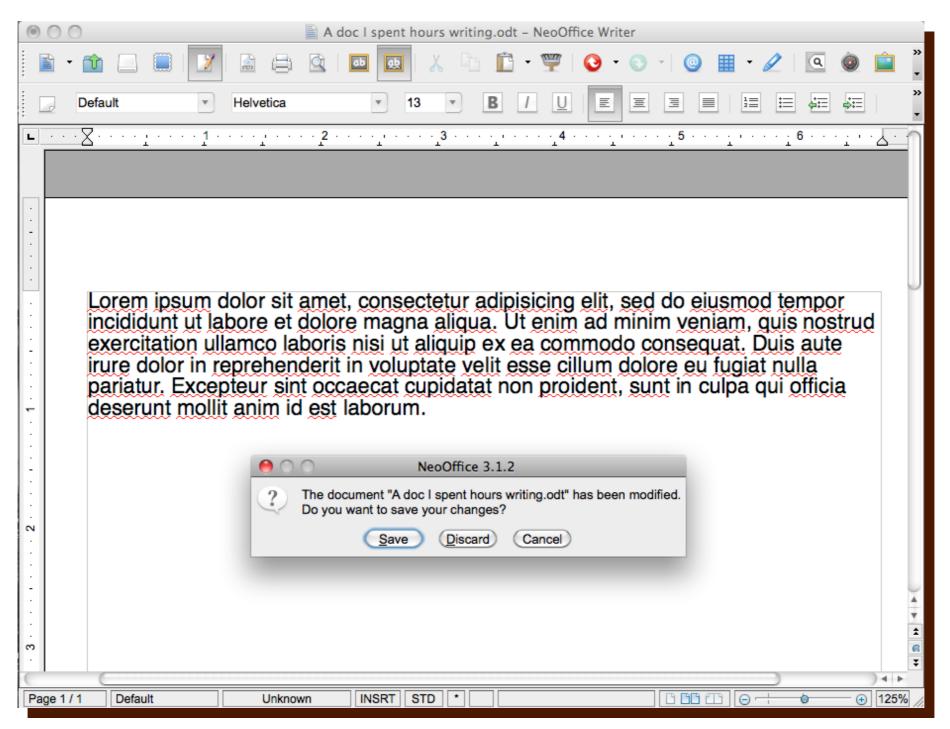
Additional

```
!/bin/bash
          Put your SLURM options here
          SBATCH --partition=debug
                                        # change to proper partition name or remove
          SBATCH --time=00:00:30
                                        # put proper time of reservation here
         #SBATCH --nodes=1
                                        # number of nodes
         #SBATCH --ntasks-per-node=1
                                        # processes per node
          SBATCH --job-name="dmtcp_job"
                                        # change to your job name
         #SBATCH --output=slurm.dmtcp
                                        # change to proper file name or remove for defaults
         #SBATCH --signal=B:USR1@60
          SBATCH --open-mode=append
 *********
 1. Start DMTCP coordinator
    *****
start_coordinator -i 10 # -i 120 ... <put dmtcp coordinator options here>
# 2. Launch application
echo "requeue #${SLURM_RESTART_COUNT}"
if [[ -e dmtcp_restart_script.sh && "${SLURM_RESTART_COUNT}" != "" ]]; then
   /bin/bash ./dmtcp_restart_script.sh -h $DMTCP_COORD_HOST -p $DMTCP_COORD_PORT &
else
     srun dmtcp_launch_-allow-file-overwrite_-rm python -u count-orig.py 10<&- 11>& &
fi
 3. setup requeue for the wall time
# Note the #SBATCH --signal=B:USR1@60 which is needed
timeout(){
echo "doing checkpoint"
dmtcp_command --checkpoint
sleep 2
echo "doing checkpoint; done"
dmtcp_command --quit
sleep 2
scontrol requeue $SLURM_JOB_ID
trap 'timeout' USR1
```



Summary, Wrap-up and Conclusions.

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