# Support for Vanilla Universe Checkpointing

Thomas Downes
University of Wisconsin-Milwaukee (LIGO)

## Experimental feature!

All features discussed are present in the official 8.5 releases.

The Morgridge Institute's Board of Ethics has decreed that these features be tested on *willing subjects only!*

## What is checkpointing?

- Saving sufficient state information to re-start execution without losing much previous work (BADPUT)
- Existing support via `condor_compile` (“standard” universe)
- Vanilla universe support: encourage jobs to periodically save sufficient state to disk and manage the migration of files

Construct policies that balance desire to minimize both BADPUT and the time to reach fair-share population of running jobs

## Why is checkpointing difficult?

- Context!
- State of process is a result of
  - explicit assumptions about its own prior actions
  - implicit assumptions about its running environment
- Fundamental problem
  - humans love context and introduce it everywhere!
  - computers… don’t

## How vanilla universe checkpointing differs

| Same as Standard Universe | Differs |
|---|---|
| • Condor daemons send a signal to request checkpoint or job can checkpoint itself<br>• Can measure success of checkpoint, time since last checkpoint, etc. | • Potentially less data transfer<br>• Greater need for users to know what they are doing<br>• Job much more likely to choose to checkpoint itself<br>• Checkpoint may occur well after signal from Condor daemon<br>• Code signals checkpoint by exiting (w/code) and restarts |

**Condor daemons should make fewer assumptions of success**

# Toy model (submit file)

```
output                     = out.log
error                      = error.log
log                        = log.log
executable                 = counting-ul
transfer_executable        = true
should_transfer_files      = true
universe                   = vanilla
transfer_input_files       = input-file
transfer_output_files      = saved-state
stream_output              = true
stream_error               = true
when_to_transfer_output    = ON_EXIT_OR_EVICT
+WantCheckpointSignal      = true
+CheckpointSig             = "SIGUSR2”
+CheckpointExitBySignal    = false
+CheckpointExitCode        = 17
+WantFTOnCheckpoint        = true
queue 1
```

Intend to support checkpoint file transfer separately from job output files!

The vanilla universe checkpoint magic

## Toy model (bash script)

```
#!/bin/bash

function PeriodicCheckpoint() {
  echo "Saving state on periodic checkpoint..."
  echo $i > saved-state
  exit 17
}

trap PeriodicCheckpoint SIGUSR2

i=0
if [ -f saved-state ]; then
  i=`cat saved-state`
fi
while [ $i != 30 ]; do
  echo $i
  sleep 60
  i=$((i+1))
done

exit 0
```

# Checkpointing real jobs

All the plumbing exists in 8.5 for you to do this, too – provide feedback to the Condor team!

## Beyond experimental

- Decided to have fun with CRIU
    - Still very experimental!
    - Key steps run as root!
    - Handy RPC interface with Python bindings
- Containers are a tool for reducing variation of job “context”
    - CRIU actively used by LXC/LXD
    - Candidate for Docker

## Set up CRIU for non-superusers

- Modify CRIU log file permissions

```
--- a/criu/log.c
+++ b/criu/log.c
- new_logfd = open(output, O_CREAT|O_TRUNC|O_WRONLY|O_APPEND, 0600);
+ new_logfd = open(output, O_CREAT|O_TRUNC|O_WRONLY|O_APPEND, 0644);
```

- Compile normally (`make && sudo make install-criu`)
- Enable dumping w/o sudo by installing on each execute node with the setuid bit

```
sudo chmod 4755 /usr/local/sbin/criu
```

- Enable restore with sudo, e.g.

```
thomas.downes ALL=(root) NOPASSWD:EXEC:/usr/local/sbin/criu
```

## Example job that checkpoints itself

```
#!/usr/bin/python

import socket, os, sys, time
import rpc_pb2 as rpc
import errno

imgdir = 'images’

s = socket.socket(socket.AF_UNIX,
socket.SOCK_SEQPACKET)
s.connect('criu_pipe')

req = rpc.criu_req()
req.type = rpc.DUMP
req.opts.leave_running = True
req.opts.shell_job = True
req.opts.evasive_devices = True
req.opts.log_file = 'test.log’
req.opts.log_level = 5
req.opts.images_dir_fd =
os.open(imgdir, os.O_DIRECTORY)
s.send(req.SerializeToString())
resp = rpc.criu_resp()
resp.ParseFromString(s.recv(1024))

if resp.success:
  print 'Checkpointed!’
else:
  print 'Epic Fail!'
```

## Writing a job that uses CRIU

- Write a wrapper
  - establishes CRIU named pipe for checkpointing operations
  - creates output directory for checkpoint images

```
[condor-test:pytest] criu service -d --address criu_pipe
[condor-test:pytest] [ -d images ] || mkdir images
[condor-test:pytest] python pytest.py
Checkpointed!
[condor-test:pytest] rm criu_pipe
[condor-test:pytest] sudo criu restore -D images –j
Checkpointed!
```

## Condor introduces context

```
[condor-test:pytest] cat important-parts-of-submit
executable                = pytest.sh
universe                  = vanilla
transfer_input_files      = pytest.py,rpc_pb2.py
transfer_output_files     = images
[condor-test:pytest] cat out.log
Checkpointed!
[condor-test:pytest] sudo criu restore -D images -j
1948: Error (files-reg.c:1524): Can't open file
var/lib/condor/execute/dir_1937/images on restore: No such file or
directory
1948: Error (files-reg.c:1466): Can't open file
var/lib/condor/execute/dir_1937/images: No such file or directory
Error (cr-restore.c:2226): Restoring FAILED.
[condor-test:pytest] sudo mkdir -p /var/lib/condor/execute/dir_17100/images
[condor-test:pytest] sudo criu restore -D images -j
### code runs however stdout has been redirected from terminal
```

## Try CRIU within Docker container!

- Create a Docker image with CRIU in it

```
[condor-test:test_image] cat Dockerfile
FROM ubuntu:16.04
ADD pytest.sh /usr/bin/pytest.sh
RUN apt-get update
RUN apt-get install --assume-yes libprotobuf-dev libprotobuf-c0-
dev protobuf-c-compiler protobuf-compiler python-protobuf libnl-
3-dev libaio-dev libcap-dev git gcc make pkg-config
RUN git clone https://github.com/xemul/criu
RUN cd criu && make && make install-criu
[condor-test:test_image] docker build –t testy .
[condor-test:pytest] cat changes-to-submit-file
universe               = docker
docker_image           = testy
```

## Oh no!

- Condor mounts the job’s unique-ish working directory to same path within the Docker container!
- Can’t be restored outside of Docker due to low PID #s (I can’t get USE_PID_NAMESPACES to work at all w/CRIU)
- But, we can play the same trick we played outside of Docker...

```
[condor-test:pytest] sudo docker run -i --privileged=true -v
/home/thomas.downes/pytest/:/var/lib/condor/execute/dir_18595 -t testy
/bin/bash
root@18e4a60da4d7:/var/lib/condor/execute/dir_18595# criu restore -D images
–j
Error (util.c:658): exec failed: No such file or directory
Error (util.c:672): exited, status=1
Error (util.c:658): exec failed: No such file or directory
Error (util.c:672): exited, status=1
```

**These error messages are red herrings. The code executes!**

## Conclusions

- Vanilla universe checkpointing management is being actively developed. *Please contribute by testing 8.5!*
- Tools like CRIU not quite ready for production, but closer every year. Condor should get ready!
- Online evidence that LXC/LXD have pulled ahead of Docker on adoption of checkpointing/migration w/CRIU.