

Self-Checkpointing Jobs with HTCondor

Todd L Miller

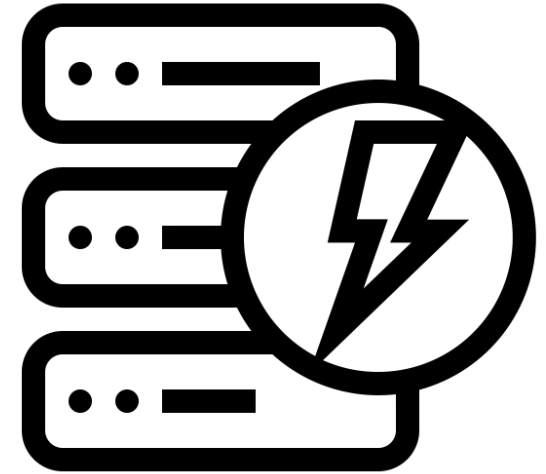
Center for High Throughput Computing

What is Checkpointing?

A program is able to save progress periodically to a file and resume from that saved file to continue running, losing minimal progress.

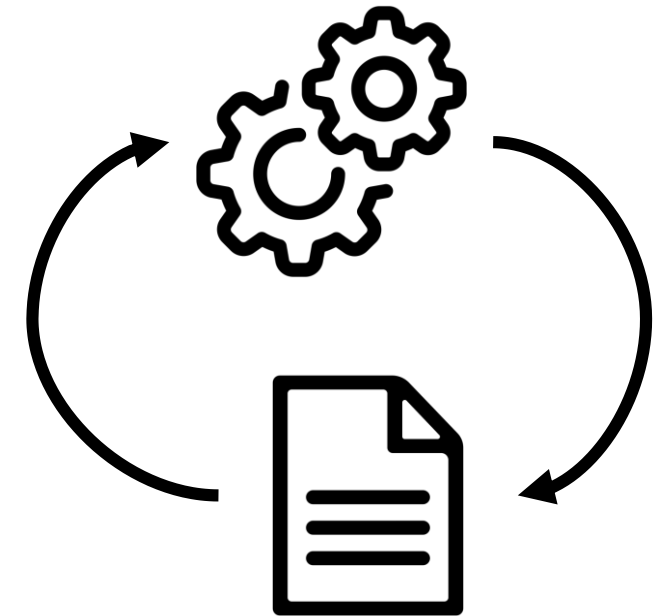
Why Self-Checkpoint?

- **Interruptions happen:**
 - Hardware or networking failures
 - Cluster/node policy (jobs can only run for 8 hours before getting killed)
 - Using opportunistic or backfill resources with no runtime guarantee
- Self-checkpointing allows you to make progress through interruptions, **especially for longer-running jobs.**



Requirements: Your Code

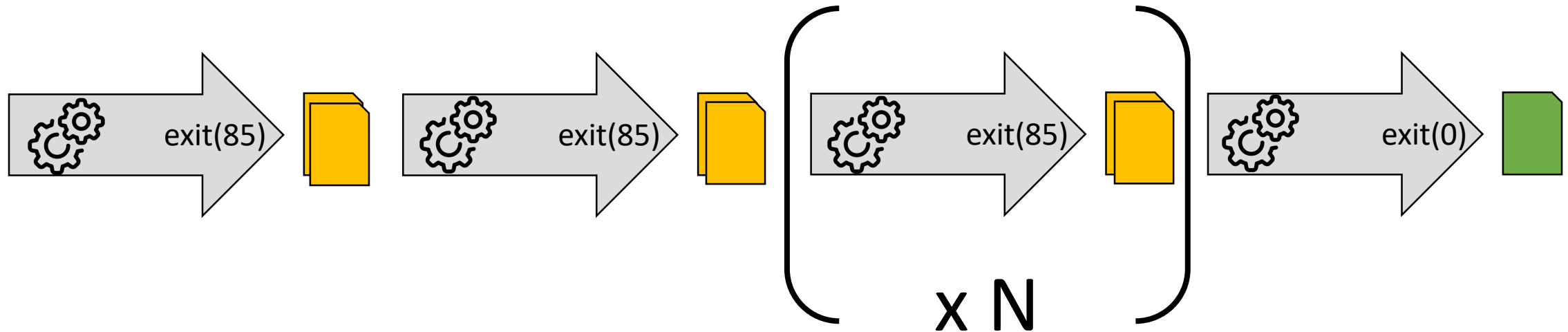
- **Ability to checkpoint and restart:**
 - *Checkpoint*: Periodically write state to a file on disk.
 - *Restart*: Code can both find the checkpoint file and can resume from it.
 - *Exit*: Code exits with a specific exit code after writing a certain number of checkpoints, but then exits with a different code after writing final output.
 - (May need a wrapper script to do some of this.)
- **Ability to checkpoint sufficiently* frequently**



Context: HTCondor

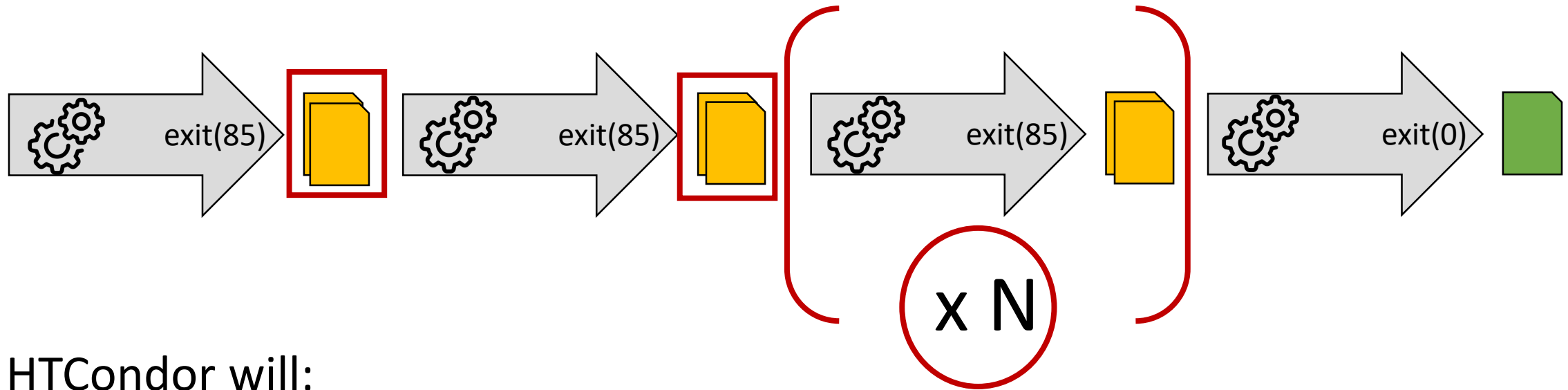
- The self-checkpointing executable is going to be run many times (to completion) by HTCondor.
- This tutorial assumes that HTCondor is managing job files (using file transfer, not a shared file system).
- This talk applies to jobs using the vanilla (or Docker) universe.
- The features discussed are first available in HTCondor 8.8.4/8.9.3; with improvements as of 9.0.6.
 - ... and bug fixes in 9.0.11, 9.3.2., 9.5.0, 9.5.1, and 9.10.0.
 - ... we're glad to see it's getting used! Thanks for finding our bugs for us.

Executable Exits After Checkpoint



- Each executable run:
 - Produces checkpoint file(s).
 - Exits with a specific code when checkpointing, and a final exit code when done.
- Note that the executable, on its own, won't run a complete execution. It needs an external process to make it repeat.

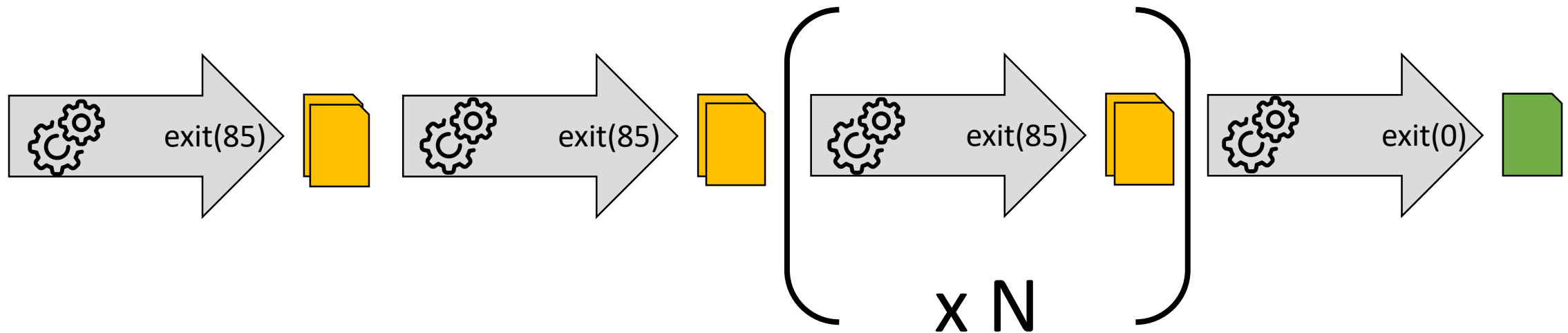
Save Checkpoint File/Resume with HTCondor





- HTCondor will:

- Restart the executable until the overall calculation is done (exit 0).
- Copy the checkpoint file(s) to a persistent location, to facilitate restarts if the job is interrupted.

Save Checkpoint File/Resume with HTCondor



```
executable = 
checkpoint_exit_code = 85
transfer_checkpoint_files = , _condor_stdout
```


Job Submitted

Submit Directory/

```
job.submit  
executable.py
```

```
job.log
```

Job Starts, Executable Starts

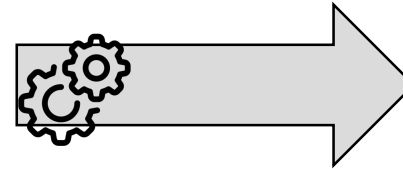
Submit Directory/

```
job.submit  
executable.py
```

```
job.log
```

Execute Node N

Execute Directory/



```
executable.py
```

```
_condor_stdout  
_condor_stderr
```

Executable Checkpoints

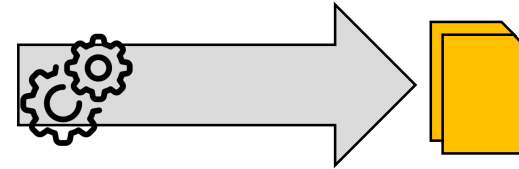
Submit Directory/

```
job.submit  
executable.py
```

```
job.log
```

Execute Node N

Execute Directory/



```
executable.py  
checkpoint.txt
```

```
_condor_stdout  
_condor_stderr
```

Executable Exits, Checkpoint Spooled

Submit Directory/

```
job.submit  
executable.py
```

```
job.log
```

Spool Directory/

```
checkpoint.txt
```

```
_condor_stdout
```

Execute Node N

Execute Directory/



```
executable.py  
checkpoint.txt
```

```
_condor_stdout  
_condor_stderr
```

Executable Started Again

Submit Directory/

```
job.submit  
executable.py
```

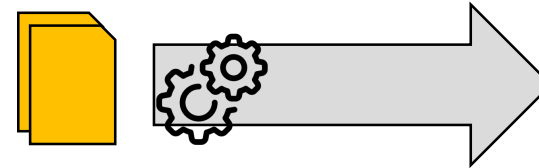
```
job.log
```

Spool Directory/

```
checkpoint.txt  
_condor_stdout
```

Execute Node N

Execute Directory/



```
executable.py  
checkpoint.txt
```

```
_condor_stdout  
_condor_stderr
```

Checkpoint Cycle Continues

Executable Interrupted

Submit Directory/

```
job.submit  
executable.py
```

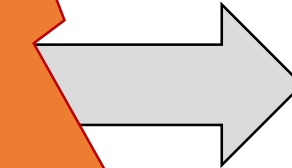
```
job.log
```

Spool Directory/

```
checkpoint.txt  
_condor_stdout
```

Execute Node N

Execute Directory/



```
executable.py  
checkpoint.txt
```

```
_condor_stdout  
_condor_stderr
```

Job Idle

Submit Directory/

```
job.submit  
executable.py
```

```
job.log
```

Spool Directory/

```
checkpoint.txt  
_condor_stdout
```


Job Restarts, Executable Restarts

Submit Directory/

job.submit
executable.py

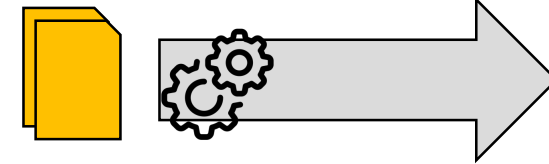
job.log

Spool Directory/

checkpoint.txt
_condor_stdout

Execute Node M

Execute Directory/



executable.py
checkpoint.txt

_condor_stdout
_condor_stderr

Checkpoint Cycle Continues

Final Execution: Executable Creates Output

Submit Directory/

```
job.submit  
executable.py
```

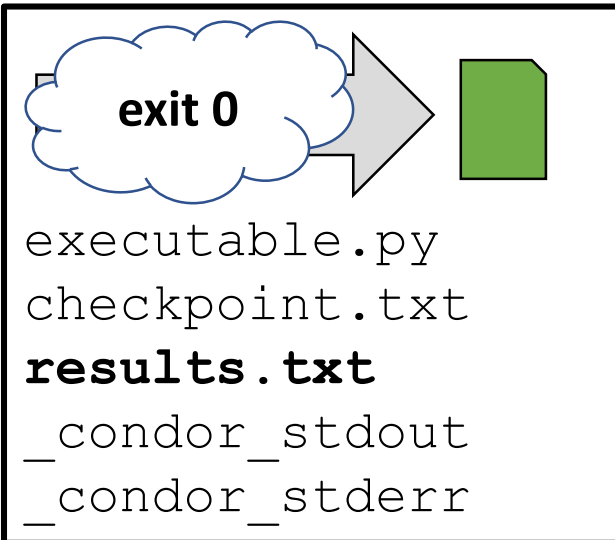
```
job.log
```

Spool Directory/

```
checkpoint.txt  
_condor_stdout
```

Execute Node M

Execute Directory/



Output Returned

Submit Directory/

```
job.submit  
executable.py
```

results.txt

```
job.log
```

```
job.out
```

```
job.err
```

Think About Output Files

- Same mechanisms for transferring output at the end of the job (triggered by executable's exit 0)
 - By default, new files are transferred back to the submission directory
 - To transfer specific output files or the contents of directories, use:
`transfer_output_files = file1, outputdir/`
- ANY output file you want to save between executable iterations (like a log file), should be included in the list of `transfer_checkpoint_files`
- This includes stdout and stderr (if using < 9.0.6.)!

Testing and Troubleshooting

- Simulate a job interruption:
 - `condor_vacate_job JobID`
- Examine your checkpoint files in the SPOOL directory:
 - Use `condor_evicted_files JobID`
 - To find the SPOOL directory: `condor_config_val SPOOL`
- Look at the HTCCondor job log for file transfer information.

Best Practices

- Scaling Up

- How many jobs will be checkpointing?
- How big are the checkpoint files?
- How much data is that total?

Avoid:

- Filling up the SPOOL directory.
- Transferring large checkpoint files.

- Checkpoint Frequency

- How long does it take to produce a checkpoint and resume?
- How likely is your job to be interrupted?

Avoid:

- Spending more time checkpointing than running.
- Jobs that will never reach a checkpoint.

Alternative Checkpointing Method

- If code can't exit after each checkpoint, but only run + checkpoint arbitrarily, transfer of checkpoint files can be triggered by eviction.
- Search for "when_to_transfer_output" on the [condor submit manual page](#); read about ON_EXIT_OR_EVICT
- This method of backing up checkpoint files is less resilient, as it won't work for other job interruption reasons (hardware issues, killed processes, held jobs).

Resources

- HTCondor Manual
 - Manual > Users' Manual > Self Checkpointing Applications
 - <https://htcondor.readthedocs.io/en/latest/users-manual/self-checkpointing-applications.html>
- Materials from the OSG Virtual School 2021
 - OSG Virtual School > Materials > Overview or Checkpointing Exercises
 - <https://opensciencegrid.org/virtual-school-2021/materials/#self-checkpointing-for-long-running-jobs>
- A longer, recorded version of this tutorial with a demo:
 - <https://videos.cern.ch/record/2782077>

Questions

Acknowledgements

This work is supported by [NSF](#) under Cooperative Agreement [OAC-2030508](#) as part of the [PATH Project](#). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.