Best Practices for HTC and Scientific Applications
Overview

1) Understand your job
2) Take it with you
3) Cache your data
4) Remote I/O
5) Be checkpointable
Understand your job

› Is it ready for HTC?
  • Runs without interaction

› Requirements are well-understood?
  • Input required
  • Execution time
  • Output generated
Understand your job

› **ALL** requirements understood?
  - Licenses
  - Network bandwidth
  - Software dependencies

› Based on the requirements, we’ll use one or more of the following strategies to get the job running smoothly
Take it with you

› Your job can run in more places, and therefore potentially access more resources, if it has fewer dependencies.
  • Don’t rely on obscure packages being installed
  • If you require a specific version of something (perl, python, etc.) consider making it part of your job
Take it with you

› Know what your set of input files is
  • Remote execution node may not share the same filesystems, and you’ll want to bring all the input with you.

› You can maybe specify the entire list of files to transfer or a directory (HTCondor)

› If the number of files is very large, but the size is small, consider creating a tarball containing the needed run-time environment
Take it with you

› Wrapper scripts can help here
  • Untar input or otherwise prepare it
  • Locate and verify dependencies
  • Set environment variables

› We use a wrapper-script approach to running Matlab and R jobs on CHTC
Take it with you

› Licensing

› Matlab requires a license to run the interpreter or the compiler, but not the results of the compilation

› Part of the submission process then is compiling the Matlab job, which is done on a dedicated, licensed machine, using HTCondor and a custom tool:
  `chtc_mcc --mfiles=my_code.m`
Another way to manage licenses is using HTCondor’s “concurrency limits”

• The user places in the submit file:
  concurrency_limits = sw_foo

• The admin places in the condor_config:
  SW_FOO_LIMIT = 10
Cache your data

› Let’s return for a moment to the compiled Matlab job
› The job still requires the Matlab runtime libraries
› As mentioned earlier, let’s not assume they will be present everywhere
Cache your data

› This runtime is the same for every Matlab job
› Running hundreds of these simultaneously will cause the same runtime to be sent from the submit node to each execute node
› CHTC solution: squid proxies
Cache your data

- The CHTC wrapper script fetches the Matlab runtime using http
- Before doing so, it also sets the http_proxy environment variable
- `curl` then automatically uses the local cache
- Can also be done with HTCondor’s file transfer plugin mechanisms, which support third party transfers (including http)
Cache your data

- The same approach would be taken for any other application that has one or more chunks of data that are “static” across jobs
  - R runtime
  - BLAST databases
Remote I/O

› What if I don’t know what data my program will access?
› Transferring everything possible may be too unwieldy and inefficient
› Consider Remote I/O
Remote I/O

› Files could be fetched on demand, again using http or whatever mechanism

› When running in HTCondor, the condor_chirp tool allows files to be fetched from and stored to during the job

› Also consider an interposition agent, such as parrot which allows trapping of I/O.
Remote I/O

› In HTCondor, add this to the submit file:
  WantRemoteIO = True

› It is off by default

› Now the job can execute:
  condor_chirp fetch /home/zmiller/foo bar
Remote I/O

- Galaxy assumes a shared filesystem for both programs and data
- Most HTCondor pools do not have this
- Initially tried to explicitly transfer all necessary files
  - This requires additional work to support each application
Remote I/O

› New approach: Parrot
  • Intercepts job’s I/O calls and redirects them back to the submitting machine
› New job wrapper for HTCondor/Parrot
  • Transfers parrot to execute machine and invokes job under parrot
› Could also be extended to have parrot do caching of large input data files
Checkpointing

› Policy on many clusters prevents jobs from running longer than several hours, or maybe up to a handful of days, before the job is preempted

› What if your job will not finish and no progress can be made?

› Make your job checkpointable
Checkpointing

› HTCondor supports “standard universe” in which you recompile (relink, actually) your executable

› Checkpoints are taken automatically when run in this mode, and when the job is rescheduled, even on a different machine, it will continue from where it left off
Checkpointing

› condor Compile is the tool used to create checkpointable jobs

› There are some limitations
  • No fork()
  • No open sockets
Checkpointing

- Condor is also working on integration with DMTCP to do checkpointing.

- Another option is user-space checkpointing. If your job can catch a signal and write its status to a file, it may be able to resume from there.
Conclusion

› Jobs have many different requirements and patterns of use
› Using one or more of the ideas above should help you get an application running smoothly on a large scale
› Questions? Please come talk to me during a break, or email zmiller@cs.wisc.edu
› Thanks!