Aalto University
School of Science

Introduction to Condor

Jari Varje

29. – 30.10.2019

Outline

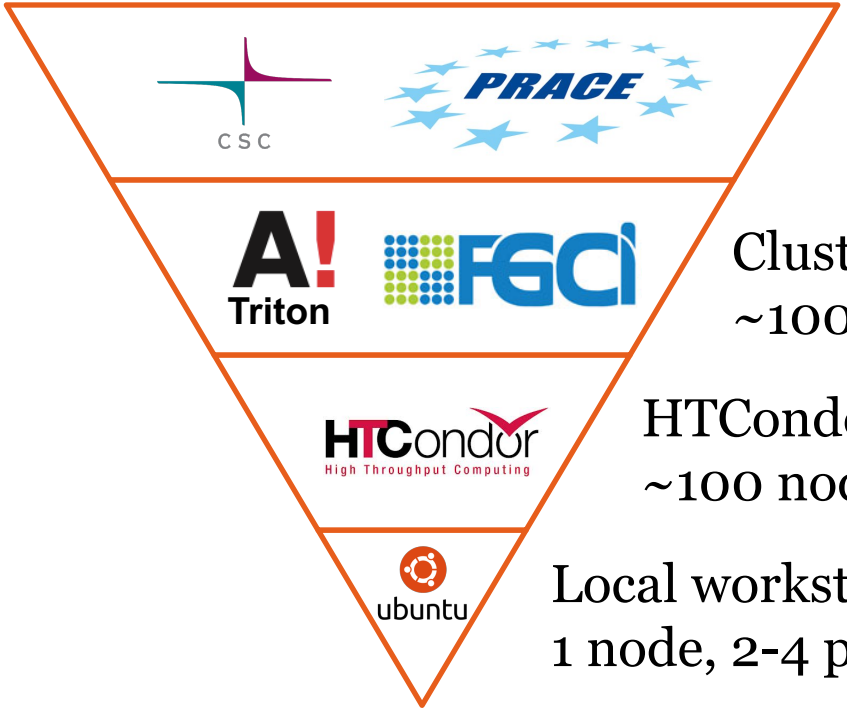
Basics

- Condor overview
- Submitting a job
- Monitoring jobs
- Parallel jobs

Advanced topics

- Host requirements
- Running MATLAB jobs
- Checkpointing
- Case study: ASCOT

Scientific computing



Supercomputing
> 1000 nodes, > 10 000 processors

Cluster, grid computing
~100-1000 nodes, ~1000-10 000 processors

HTCondor
~100 nodes, ~500 processors

Local workstation
1 node, 2-4 processors

High Performance vs High Throughput Computing

High Performance Computing (HPC)

- Simultaneous execution of tightly coupled tasks
- Computationally difficult problems
- Massively parallel, interconnected systems
- e.g. Triton, Sisu

High Performance vs High Throughput Computing

High Performance Computing (HPC)

- Simultaneous execution of tightly coupled tasks
- Computationally difficult problems
- Massively parallel, interconnected systems
- e.g. Triton, Sisu

High Throughput Computing (HTC)

- Large number of loosely coupled tasks processed over time
- Data analysis, Monte Carlo simulations
- Distributed systems
- e.g. BOINC (Seti@home), Condor

Condor overview

HTCondor - Framework for High Throughput Computing
https://research.cs.wisc.edu/htcondor/

- Schedules and coordinates job execution on computing resources
- Dedicated clusters
- Heterogeneous clusters
- Grid computing
- CPU scavenging

Condor overview

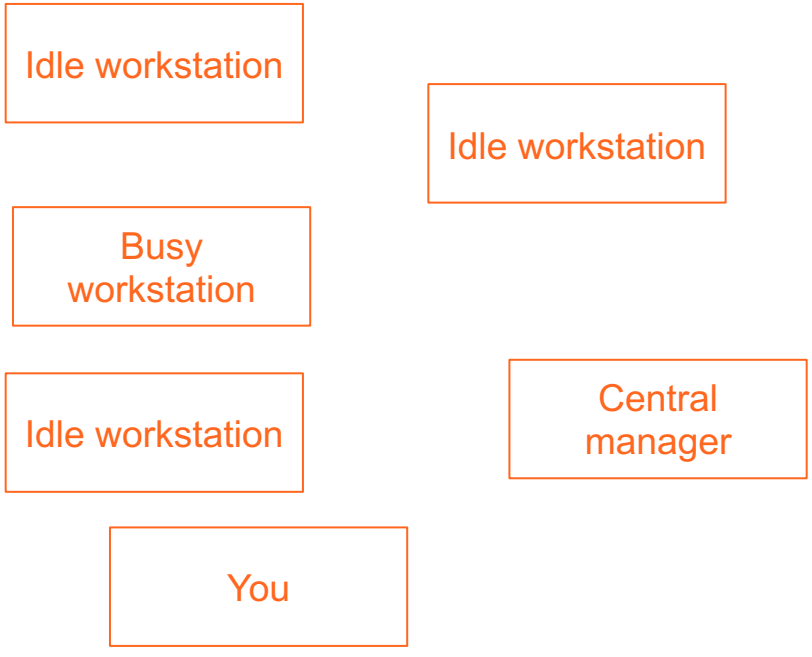
HTCondor - Framework for High Throughput Computing
https://research.cs.wisc.edu/htcondor/

- Schedules and coordinates job execution on computing resources
- Dedicated clusters
- Heterogeneous clusters
- Grid computing
- CPU scavenging

For us:

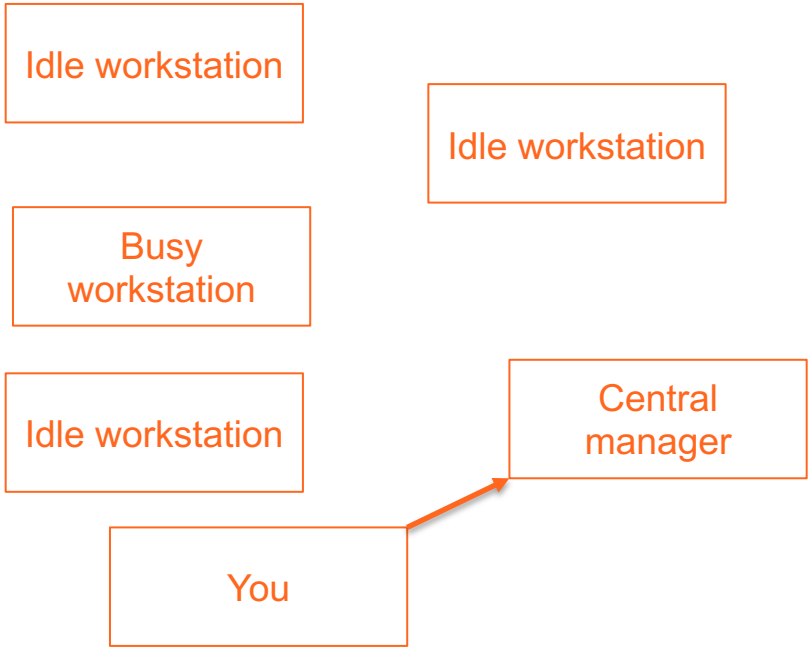
- System for executing codes on idle workstations

Condor overview



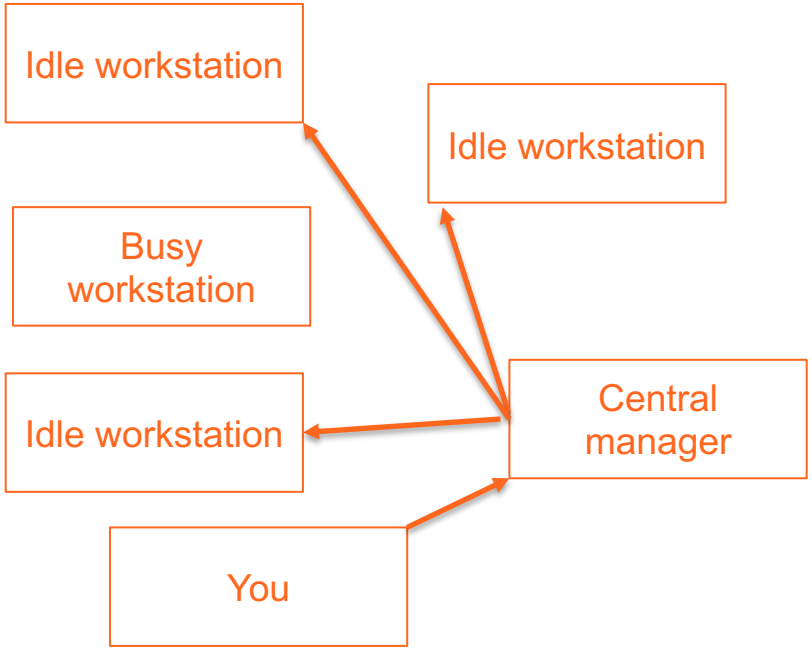
Condor overview

1. Submit job to Condor



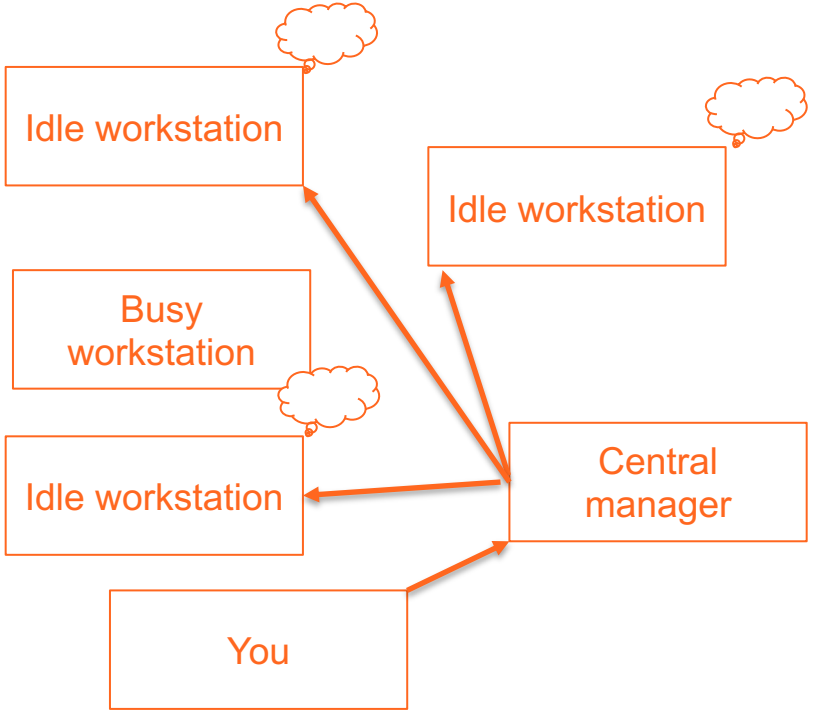
Condor overview

1. Submit jobs to Condor
2. Central manager schedules the jobs to idle hosts
3. Code and input files transferred to hosts



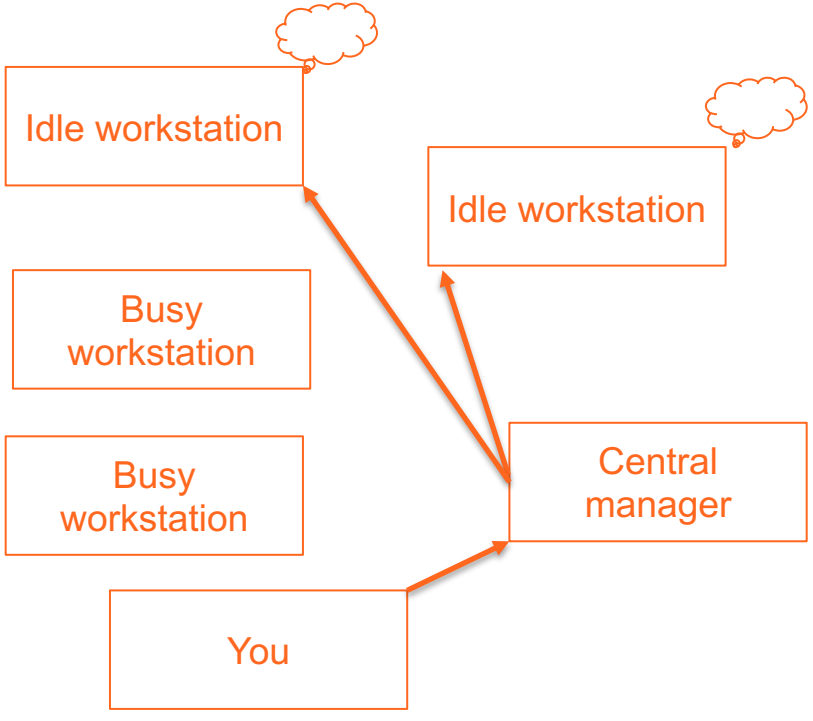
Condor overview

1. Submit jobs to Condor
2. Central manager schedules the jobs to idle hosts
3. Code and input files transferred to hosts
4. Code executes until complete…



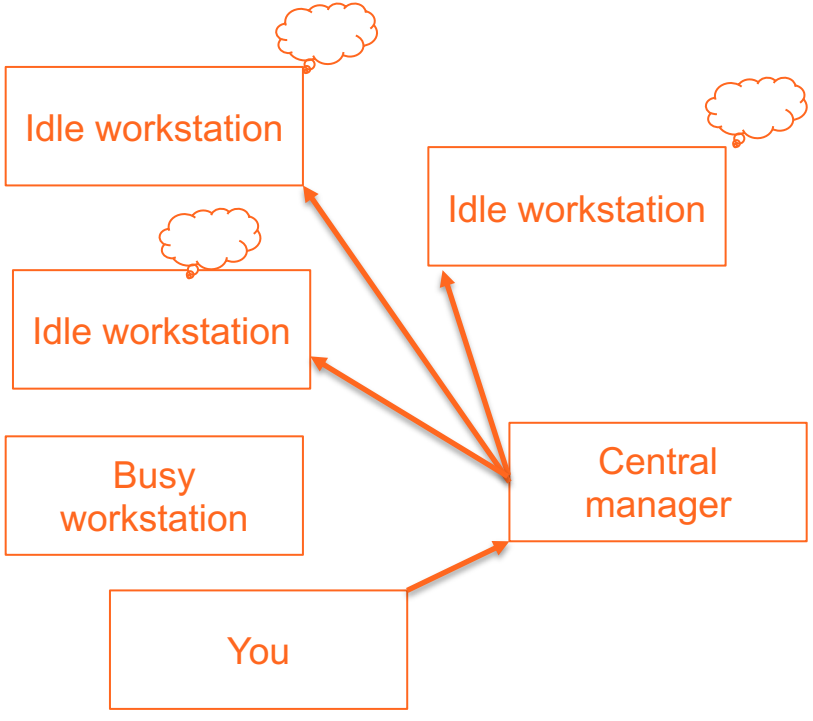
Condor overview

1. Submit jobs to Condor
2. Central manager schedules the jobs to idle hosts
3. Code and input files transferred to hosts
4. Code executes until complete… or evicted…



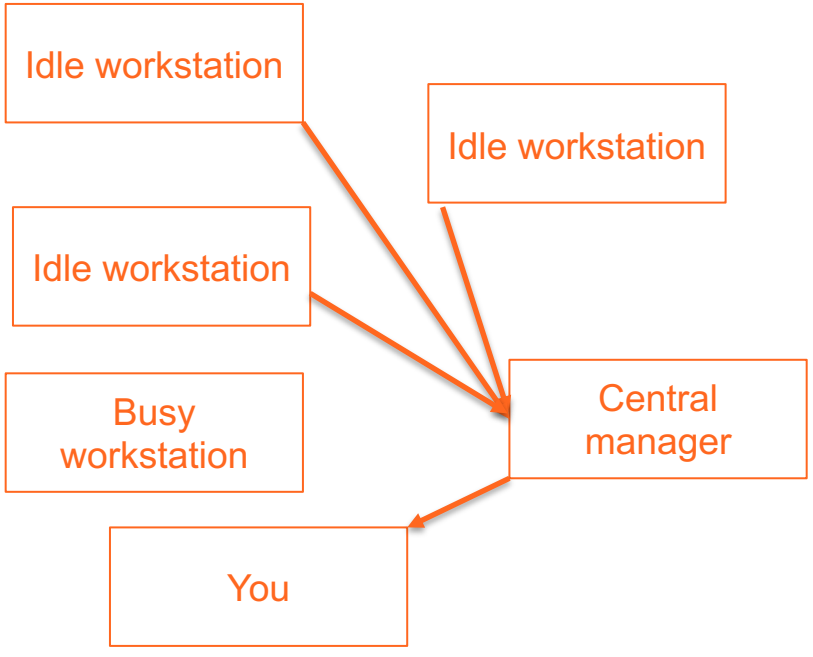
Condor overview

1. Submit jobs to Condor
2. Central manager schedules the jobs to idle hosts
3. Code and input files transferred to hosts
4. Code executes until complete… or evicted… and rescheduled



Condor overview

1. Submit jobs to Condor
2. Central manager schedules the jobs to idle hosts
3. Code and input files transferred to hosts
4. Code executes until complete… or evicted… and rescheduled
5. Output files transferred back to submitter



What kind of jobs?

- Individual serial jobs
- Independent parallel jobs – no communication!

- Limited memory
- Limited I/O – data transferred individually to hosts
- Limited parallelization
- Reasonable execution duration ~1-12 h

Jobs as clusters and processes

- Each submitted job identified by *ClusterId*
  - Unique to submitting machine
- Each process of the job identified by *ProcId*

```
ClusterId.ProcId

1.0

2.0

3.0
3.1
3.2
...
3.100
```

Condor commands

condor_submit	Submit a job
condor_q	Show job queue
condor_hold / condor_release	Hold and continue execution of a job
condor_rm	Cancel a job
condor_status	Status of the Condor cluster
condor_history	Completed jobs
condor_prio	Increase/decrease job priority
condor_qedit	Change job parameters
condor_userprio	Condor usage statistics / accounting

Example 1: Serial job

- Job submitted with a submit description file
  - Executable file (transferred automatically)
  - Output: stdout, stderr, logging
  - *Queue* command

script1.py

```
#!/usr/bin/python
import time
print "Starting“
time.sleep(60)
print "Finishing"
```

script1.cmd

```
Executable = script1.py
Output = condor.out
Error = condor.err
Log = condor.log
Should_transfer_files = Yes
Queue
```

Example 1: Serial job

```
$ condor_status
Name               OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

slot1@andy.hut.fi  LINUX      X86_64 Owner     Idle      0.000 2992  0+01:15:04
slot2@andy.hut.fi  LINUX      X86_64 Owner     Idle      0.000 2992  0+01:15:05
slot3@andy.hut.fi  LINUX      X86_64 Owner     Idle      0.060 2992  0+01:15:06
slot4@andy.hut.fi  LINUX      X86_64 Owner     Idle      0.000 2992  0+01:15:07
slot1@anitra.hut.f LINUX      X86_64 Unclaimed Idle      0.100 2950  0+00:15:04
slot2@anitra.hut.f LINUX      X86_64 Unclaimed Idle      0.000 2950  0+14:45:08
slot3@anitra.hut.f LINUX      X86_64 Unclaimed Idle      0.000 2950  0+14:45:09
slot4@anitra.hut.f LINUX      X86_64 Unclaimed Idle      0.000 2950  0+14:45:10
...
                     Total Owner Claimed Unclaimed Matched Preempting Backfill

        X86_64/LINUX   321   170       0       151       0          0        0

               Total   321   170       0       151       0          0        0
```

Example 1: Serial job

```
$ condor_submit script1.cmd
Submitting job(s).
1 job(s) submitted to cluster 261.
$ condor_q
-- Submitter: nauris.hut.fi : <130.233.204.143:9618?sock=1395_7d4e_4> : nauris.hut.fi
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 261.0   jvarje          4/25 11:14   0+00:00:02 R  0   0.0  script1.py

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
$ condor_history
 ID     OWNER          SUBMITTED   RUN_TIME     ST COMPLETED   CMD
 261.0   jvarje          4/25 11:14   0+00:01:00 C   4/25 11:15 /home/jvarje/co
...
```

Example 2: Parallel jobs

script2.py

```
#!/usr/bin/python
import time
import sys
print "Starting process", sys.argv[2], "of", sys.argv[1]
time.sleep(60)
print "Finishing process", sys.argv[2], "of", sys.argv[1]
```

```
./script2.py 10 0
```

Example 2: Parallel jobs

- Job identity in command line arguments
- Condor variables
  - *$(Cluster)* – id for this run
  - *$(Process)* – id for the specific job in the cluster
- User variables
  - *$(My_processes)* – number of parallel processes

script2.cmd

```
My_processes = 10

Executable = script2.py
Arguments = $(My_processes) $(Process)

Output = condor.out.$(Cluster).$(Process)
Error = condor.err.$(Cluster).$(Process)
Log = condor.log.$(Cluster).$(Process)

Should_transfer_files = Yes

Queue $(My_processes)
```

Example 2: Parallel jobs

```
$ condor_submit script2.cmd
Submitting job(s)..........
10 job(s) submitted to cluster 262.
$ condor_q
-- Submitter: nauris.hut.fi : <130.233.204.143:9618?sock=1395_7d4e_4> : nauris.hut.fi
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 262.0   jvarje          4/25 11:23   0+00:00:00 I  0   0.0  script2.py 10 0
 262.1   jvarje          4/25 11:23   0+00:00:00 I  0   0.0  script2.py 10 1
...
 262.9   jvarje          4/25 11:23   0+00:00:00 I  0   0.0  script2.py 10 9

11 jobs; 0 completed, 0 removed, 11 idle, 0 running, 0 held, 0 suspended
$ condor_q 262.7
-- Submitter: nauris.hut.fi : <130.233.204.143:9618?sock=1395_7d4e_4> : nauris.hut.fi
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 262.7   jvarje          4/25 11:23   0+00:00:40 R  0   0.0  script2.py 10 7
$
```

Example 3: Job with output files

script3.py

```
#!/usr/bin/python
import time
import sys
print "Starting process", sys.argv[2], "of", sys.argv[1]
time.sleep(60)
f=open("input.dat", "r")
lines=f.readlines()
f.close()
f=open("output." + sys.argv[2] + ".dat", "w")
f.write("Process " + sys.argv[2] + " of " + sys.argv[1] + "\n")
f.writelines(lines)
f.close()
print "Finishing process", sys.argv[2], "of", sys.argv[1]
```

Example 3: Job with output files

- Executable transferred automatically
- Input files must be specified separately
- All output files transferred back by default
  - Can be restricted with *Transfer_output_files*

script3.cmd

```
My_processes = 10

Executable = script3.py
Arguments = $(My_processes) $(Process)

Output = condor.out.$(Cluster).$(Process)
Error = condor.err.$(Cluster).$(Process)
Log = condor.log.$(Cluster).$(Process)

Should_transfer_files = Yes
Transfer_input_files = input.dat
#Transfer_output_files = output.dat

Queue $(My_processes)
```

Example 3: Job with output files

- Executable transferred automatically
- Input files must be specified separately
- All output files transferred back by default
  - Can be restricted with *Transfer_output_files*
- Transfer files through Condor!

script3.cmd

```
My_processes = 10

Executable = script3.py
Arguments = $(My_processes) $(Process)

Output = condor.out.$(Cluster).$(Process)
Error = condor.err.$(Cluster).$(Process)
Log = condor.log.$(Cluster).$(Process)

Should_transfer_files = Yes
Transfer_input_files = input.dat
#Transfer_output_files = output.dat

Queue $(My_processes)
```

Example 3: Job with output files

```
$ condor_submit script3.cmd
Submitting job(s)..........
10 job(s) submitted to cluster 263.
$ condor_hold 263
Cluster 263 held.
$ condor_q
-- Submitter: nauris.hut.fi : <130.233.204.143:9618?sock=1395_7d4e_4> : nauris.hut.fi
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
 263.0   jvarje          4/25 11:35   0+00:00:00 H  0   0.0  script3.py 10 0
...
 263.9   jvarje          4/25 11:35   0+00:00:00 H  0   0.0  script3.py 10 9

10 jobs; 0 completed, 0 removed, 0 idle, 0 running, 10 held, 0 suspended
$ condor_release 263.7
Job 263.7 released
$ condor_rm 263
Cluster 263 has been marked for removal.
$
```

Outline

Basics

- Condor overview
- Submitting a job
- Monitoring jobs
- Parallel jobs

Advanced topics

- Host requirements
- Running MATLAB jobs
- Checkpointing
- Case study: ASCOT

Host requirements

- Condor supports heterogeneous systems – resources available per node can vary widely
  - Number of CPUs
  - Memory
  - Disk space
  - Architecture (x86, x86_64…)
  - Operating System (Linux, Windows…)
  - OS version

Example 4a: Host requirements

- Any host parameters can be used as a requirement
- Resource requests automatically appended to Requirements
- Preferences can be passed to scheduler with Rank

script4a.cmd

```
Executable = script4a.py
Output = condor.out
Error = condor.err
Log = condor.log

Requirements = OpSys == "LINUX" && Arch ==
                                                "X86_64”
#Requirements = OpSysVer == 1404

request_memory = 100 GB
request_cpus = 1
request_disk = 100 MB
rank = Memory

Queue
```

Example 4b: GPUs

- Any host parameters can be used as a requirement
- Resource requests automatically appended to Requirements
- Preferences can be passed to scheduler with Rank

script4b.cmd

```
Executable = script4b
Output = condor.out
Error = condor.err
Log = condor.log

request_GPUs = 1
#requirements = CudaCapability >= 3.0

Queue
```

Example 5a: MATLAB job

- Simple MATLAB jobs can be executed as scripts
- *Exit* the script to avoid hanging
- MATLAB environment set up in a wrapper script

script5a.m

```
disp('Starting');
pause(60);
disp('Finishing');
exit
```

script5a.sh

```
#!/bin/bash –login
module load matlab
matlab
```

Example 5a: MATLAB job

- Simple MATLAB jobs can be executed as scripts
- *Exit* the script to avoid hanging
- MATLAB environment set up in a wrapper script
- Script passed to standard input
- Only wrapper script transferred automatically!

script5a.cmd

```
Executable = script5a.sh

Output = condor.out
Error = condor.err
Log = condor.log
Input = script5a.m

Should_transfer_files = Yes
Transfer_input_files = script5a.m

Queue
```

Example 5b: MATLAB job

- More complicated MATLAB codes as functions with parameters
- Parameters passed with command line arguments

script5b.m

```
function script5b(numprocesses, iprocess)
 disp(['Starting process ',num2str(iprocess),'
of ',num2str(numprocesses)]);

 pause(60);

 disp(['Finishing process ',num2str(iprocess),'
of ',num2str(numprocesses)]);
end
```

script5b.sh

```
#!/bin/bash
matlab -r "script5b($1,$2)"
```

Example 5b: MATLAB job

- More complicated MATLAB codes as functions with parameters
- Parameters passed with command line arguments
- Environment can be set at submission time with Getenv

script5b.cmd

```
My_processes = 1

Executable = script5b.sh
Arguments = $(My_processes) $(Process)
Getenv = True


Output = condor.out.$(Cluster).$(Process)
Error = condor.err.$(Cluster).$(Process)
Log = condor.log.$(Cluster).$(Process)


Should_transfer_files = Yes
Transfer_input_files = script5b.m


Queue $(My_processes)
```

Checkpointing

- Job progress lost if process is evicted
- Solution: store computation state by checkpointing
- Condor supports automatic checkpointing
  - Entire process state stored in a checkpoint repository upon eviction
  - Process resumes as if nothing happened
  - Execute in *Standard* Universe (*Vanilla* by default)

Example 6a: Checkpointing

- No changes to code – Recompile using *condor_compile*
- Supports C/C++, Fortran

script6a.c

```
#include <stdio.h>
#include <unistd.h>
int main() {
    int i;
    for(i = 0; i < 60; i++) {
        printf("%d\n", i);
        sleep(1);
    }
    return 0;
}
```

```
condor_compile gcc –o script6a script6a.c
```

Example 6a: Checkpointing

- No changes to code – Recompile using *condor_compile*
- Supports C/C++, Fortran
- But not supported in our Condor (yet?)
- Workarounds:
  - Checkpoint at regular intervals
  - Catch termination signal

script6a.cmd

```
Universe = Standard
Executable = script6a

Output = condor.out
Error = condor.err
Log = condor.log

Queue
```

Example 6b: Regular checkpointing

- Relevant process state manually written at regular intervals
- Inefficient with large data sets

script6b.py

```
#!/usr/bin/python
import time, os
if(os.path.exists('checkpoint')):
    f = open('checkpoint','r')
    start = int(f.read()) + 1
    f.close()
else:
    start = 1

print "Starting”
for i in range(start,60):
    print i
    f = open('checkpoint','w')
    f.write(str(i))
    f.close()
    time.sleep(1)
print "Finishing"
```

Example 6b: Regular checkpointing

- Relevant process state manually written at regular intervals
- Inefficient with large data sets
- Checkpoint file transferred back upon eviction

script6b.cmd

```
Executable = script6b.py

Output = condor.out
Error = condor.err
Log = condor.log

skip_filechecks = true
transfer_input_files = checkpoint
when_to_transfer_output = ON_EXIT_OR_EVICT

Queue
```

Example 6c: Checkpoint at signal

- Condor sends SIGTERM signal upon eviction
- Limited time to write checkpoint before SIGKILL

script6c.py

```
#!/usr/bin/python
import time, os, signal
if(os.path.exists('checkpoint')):
    f = open('checkpoint','r')
    start = int(f.read()) + 1
    f.close()
else:
    start = 1

def handler(signum, frame):
    f = open('checkpoint','w')
    f.write(str(i))
    f.close()
    exit(1)
signal.signal(signal.SIGTERM, handler)

print "Starting”
for i in range(start,60):
    print i
    time.sleep(1)
print "Finishing"
```

Case study: ASCOT

Plasma kinetic equation solver

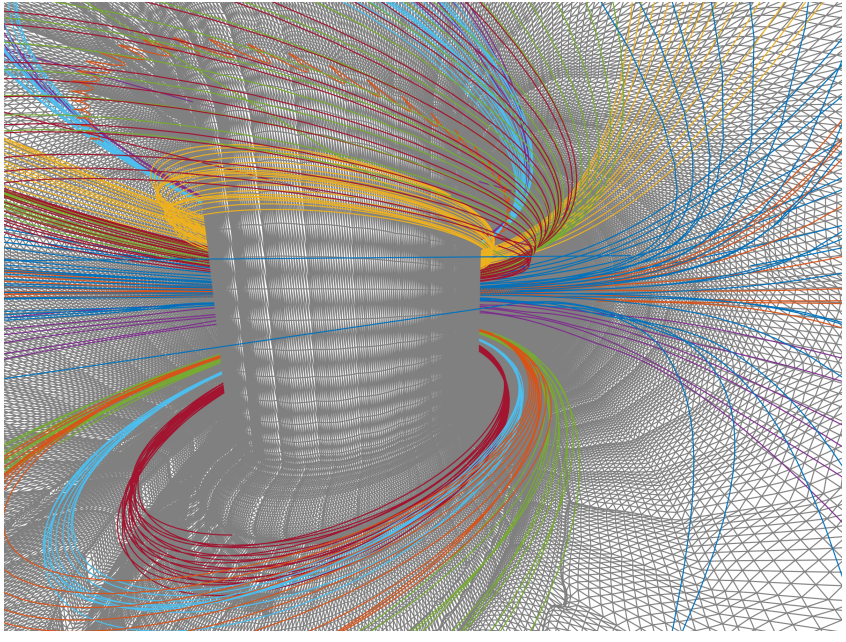
- Monte Carlo orbit following

Massively parallel

- 8000-16000 cores

Embarrassingly parallel

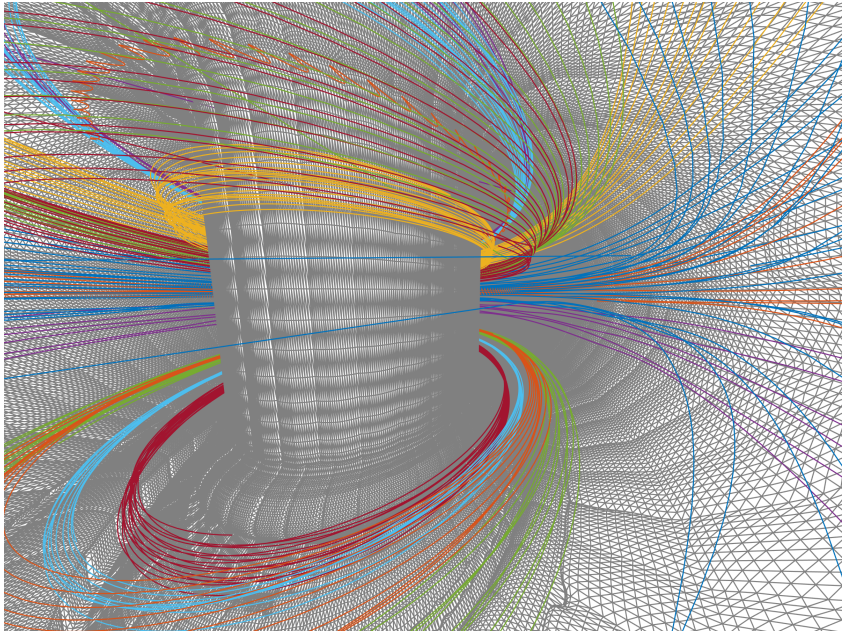
- Each particle followed independently (typically)



Case study: ASCOT

ASCOT4

- Fortran 90
- Single-threaded
- Parallelized using MPI



Case study: ASCOT

- MPI calls replaced with dummy functions[1] when compiling for Condor
- MPI size, rank set by command line arguments
- Inert communication functions

Makefile

```
...
ifdef NO_MPI
  MPIFC=$(FC)
  MPI_OBJS = mpi_stubs.o
endif
...
```

mpi_stubs.f90

```
...
subroutine mpi_comm_rank(comm, rank, ierror)
  use mpivar, only : mpivar_rank
  rank=mpivar_rank
  ierror=MPI_SUCCESS
end subroutine mpi_comm_rank
...
```

[1] http://people.sc.fsu.edu/~jburkardt/f_src/mpi_stubs/mpi_stubs.html

Case study: ASCOT

- MPI_gather etc. only copy local data
- Each process stores their results independently

```
...
call gatherStore(endstate,endstateGathered)
write(fileName,'(A,I6.6,A)') &
     'ascot_output_',mpirank,'.h5‘
if(mpirank==0 .or.  &
     mpivar_independentParallelJob) then
  call writeEndState( &
     endstateGathered,TRIM(fileName))
end
...
```

Case study: ASCOT

```
NUMPROC = 1000
Universe = Vanilla

Executable = ./test_ascot
Arguments = -mpi_size $(NUMPROC) -mpi_rank $(Process) -output ascot_fcjvar_$(cluster)

Output = condor.out.$(cluster).$(Process)
Error =  condor.err.$(cluster).$(Process)
Log =    condor.log.$(cluster).$(Process)

transfer_input_files =  input.magn_bkg, input.magn_header, input.options, input.particles,
input.plasma_1d, input.wall_2d
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
on_exit_remove = (ExitBySignal == False) && (ExitCode == 0)

Rank = kflops
Requirements = (TotalFreeMemory >= 1000) && ( MACHINE != "art.physics.aalto.fi" )

Queue $(NUMPROC)
```