Introduction to Condor

based on material by Miron Livny et al

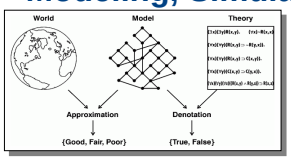
http://www.cs.wisc.edu/condor

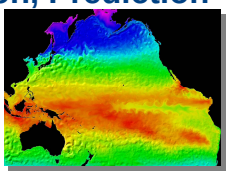
Motivation, Overview

The Story of Frieda, the Scientist

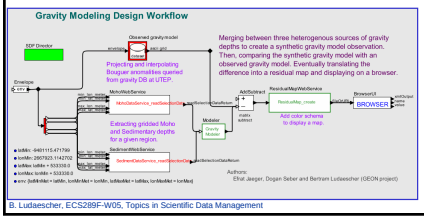
- Using Condor to manage jobs
- Using Condor to manage resources
- Condor Architecture and Mechanisms
- Condor on the Grid
  - Flocking
  - Condor-G

Modeling, Simulation, Prediction

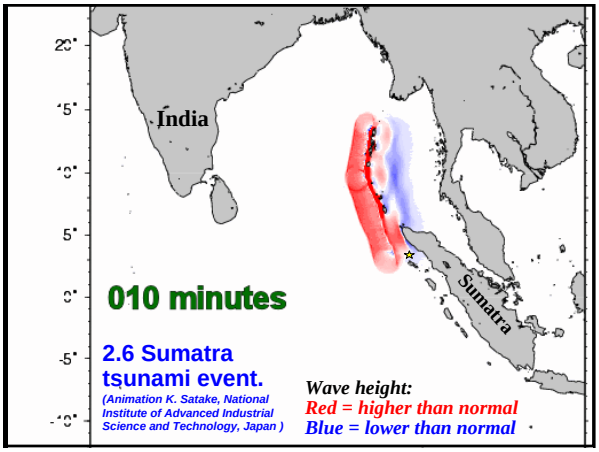




"All models are wrong, but some are useful!"



Snapshot of sea surface temperature (shown in color) in shaded relief format using sea level slope simulated by the 12.5-km ROMS over the Pacific Ocean. (Source: Yi Chao, Jet Propulsion Lab)



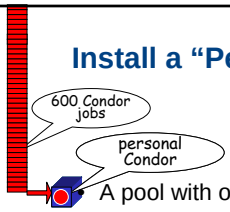
Back to CS: Frieda's Application …

Meet Frieda: She is a scientist.
But she has a big problem: I have 600 simulations to run.
Where can I get help?

Simulate the behavior of **F**(*x*,*y*,*z*) for 20 values of *x*, 10 values of *y* and 3 values of *z* (20*10*3 = 600 combinations)

- **F** takes on the average 6 hours to compute on a "typical" workstation (total = 1800 hours)
- **F** requires a "moderate" (128MB) amount of memory
- **F** performs "moderate" I/O - (*x*,*y*,*z*) is 5 MB and **F**(*x*,*y*,*z*) is 50 MB

Install a "Personal Condor Pool"



- A pool with one node … !??
- Benefits:
  - … keep an eye on your jobs and will keep you posted on their progress
  - … implement your policy on the execution order of the jobs
  - … keep a log of your job activities
  - … add fault tolerance to your jobs
  - … implement your policy on when the jobs can run on your workstation

Getting Started: Submitting Jobs to Condor

- Choosing a "Universe" (runtime environment) for your job
  - Just use VANILLA for now
- Make your job "batch-ready"
- Creating a *submit description* file
- Run *condor_submit* on your submit description file

Making your job batch-ready

- Must be able to run in the background: no interactive input, windows, GUI, etc.
- Can still use `STDIN`, `STDOUT`, and `STDERR` (the keyboard and the screen), but files are used for these instead of the actual devices
- Organize data files

Creating a Submit Description File

- A plain ASCII text file
- Tells Condor about your job:
  - Which executable, universe, input, output and error files to use, command-line arguments, environment variables, any special requirements or preferences (more on this later)
- Can describe many jobs at once (a "cluster") each with different input, arguments, output, etc.

Simple Submit Description File

```
# Simple condor_submit input file
# (Lines beginning with # are comments)
# NOTE: the words on the left side are not
#       case sensitive, but filenames are!
Universe   = vanilla
Executable = my_job
Queue
```


Running condor_submit

- You give *condor_submit* the name of the submit file you have created
- *condor_submit* parses the file, checks for errors, and creates a "ClassAd" that describes your job(s)
- Sends your job's ClassAd(s) and executable to the condor_schedd, which stores the job in its queue
  - Atomic operation, two-phase commit
- View the queue with *condor_q*

Running condor_submit

```
% condor_submit my_job.submit-file
Submitting job(s).
1 job(s) submitted to cluster 1.

% condor_q

-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  1.0    frieda           6/16 06:52   0+00:00:00 I  0   0.0  my_job

1 jobs; 1 idle, 0 running, 0 held

%
```


Another Submit Description File

```
# Example condor_submit input file
# (Lines beginning with # are comments)
# NOTE: the words on the left side are not
#       case sensitive, but filenames are!
Universe   = vanilla
Executable = /home/wright/condor/my_job.condor
Input      = my_job.stdin
Output     = my_job.stdout
Error      = my_job.stderr
Arguments  = -arg1 -arg2
InitialDir = /home/wright/condor/run_1
Queue
```

"Clusters" and "Processes"

- If your submit file describes multiple jobs, we call this a "cluster"
- Each job within a cluster is called a "process" or "proc"
- If you only specify one job, you still get a cluster, but it has only one process
- A Condor "Job ID" is the cluster number, a period, and the process number ("23.5")
- Process numbers always start at 0

Example Submit Description File for a Cluster

```
# Example condor_submit input file that defines
# a cluster of two jobs with different iwd
Universe   = vanilla
Executable = my_job
Arguments  = -arg1 -arg2

InitialDir = run_0
Queue      ← Becomes job 2.0

InitialDir = run_1
Queue      ← Becomes job 2.1
```



```
% condor_submit my_job.submit-file
Submitting job(s).
2 job(s) submitted to cluster 2.

% condor_q

-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  1.0    frieda           6/16 06:52  0+00:02:11 R  0   0.0  my_job
  2.0    frieda           6/16 06:56  0+00:00:00 I  0   0.0  my_job
  2.1    frieda           6/16 06:56  0+00:00:00 I  0   0.0  my_job

3 jobs; 2 idle, 1 running, 0 held

%
```


Submit Description File for a *BIG* Cluster of Jobs

- The initial directory for each job is specified with the $(Process) macro, and instead of submitting a single job, we use "Queue 600" to submit 600 jobs at once
- $(Process) will be expanded to the process number for each job in the cluster (from 0 up to 599 in this case), so we'll have "run_0", "run_1", … "run_599" directories
- All the input/output files will be in different directories!

Submit Description File for a *BIG* Cluster of Jobs

```
# Example condor_submit input file that defines
# a cluster of 600 jobs with different iwd
Universe   = vanilla
Executable = my_job
Arguments  = -arg1 –arg2
InitialDir = run_$(Process)
Queue 600
```


Using condor_rm

- If you want to remove a job from the Condor queue, you use *condor_rm*
- You can only remove jobs that you own (you can't run *condor_rm* on someone else's jobs unless you are root)
- You can give specific job ID's (cluster or cluster.proc), or you can remove all of your jobs with the "*-a*" option.

Temporarily halt a Job

- Use *condor_hold* to place a job on hold
  - Kills job if currently running
  - Will not attempt to restart job until released
  - Sometimes Condor will place a job on hold ("system hold")
- Use *condor_release* to remove a hold and permit job to be scheduled again

Using condor_history

- Once your job completes, it will no longer show up in *condor_q*
- You can use *condor_history* to view information about a completed job
- The status field ("ST") will have either a "C" for "completed", or an "X" if the job was removed with *condor_rm*

Getting Email from Condor

- By default, Condor will send you email when your jobs completes
  - With lots of information about the run
- If you don't want this email, put this in your submit file:

```
notification = never
```

- If you want email every time something happens to your job (preempt, exit, etc), use this:

```
notification = always
```


Getting Email from Condor (cont'd)

- If you only want email in case of errors, use this:

```
notification = error
```

- By default, the email is sent to your account on the host you submitted from. If you want the email to go to a different address, use this:

```
notify_user = email@address.here
```


A Job's life story: The "User Log" file

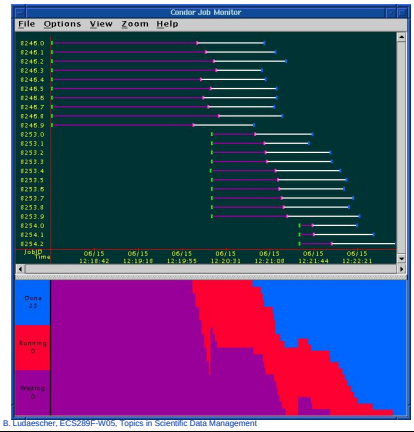
- A UserLog must be specified in your submit file:
  - Log = filename
- You get a log entry for everything that happens to your job:
  - When it was submitted, when it starts executing, preempted, restarted, completes, if there are any problems, etc.
- Very useful! Highly recommended!

Sample Condor User Log

```
000 (8135.000.000) 05/25 19:10:03 Job submitted from host: <128.105.146.14:1816>
...
001 (8135.000.000) 05/25 19:12:17 Job executing on host: <128.105.165.131:1026>
...
005 (8135.000.000) 05/25 19:13:06 Job terminated.
        (1) Normal termination (return value 0)
                Usr 0 00:00:37, Sys 0 00:00:00  -  Run Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:05  -  Run Local Usage
                Usr 0 00:00:37, Sys 0 00:00:00  -  Total Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:05  -  Total Local Usage
        9624  -  Run Bytes Sent By Job
        7146159  -  Run Bytes Received By Job
        9624  -  Total Bytes Sent By Job
        7146159  -  Total Bytes Received By Job
...
```

Uses for the User Log

- Easily read by human or machine
  - C++ library and Perl Module for parsing UserLogs is available
- Event triggers for meta-schedulers
  - Like DagMan…
- Visualizations of job progress
  - Condor JobMonitor Viewer



Condor JobMonitor Screenshot

Job Priorities w/ condor_prio

- *condor_prio* allows you to specify the order in which your jobs are started
- Higher the prio #, the earlier the job will start

```
% condor_q
-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  1.0    frieda          6/16 06:52  0+00:02:11 R  0   0.0  my_job
% condor_prio +5 1.0
% condor_q
-- Submitter: perdita.cs.wisc.edu : <128.105.165.34:1027> :
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  1.0    frieda          6/16 06:52  0+00:02:13 R  5   0.0  my_job
```


Want other Scheduling possibilities? Use the Scheduler Universe

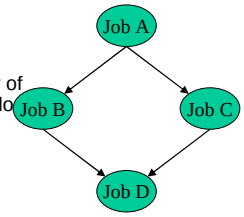
- In addition to VANILLA, another job universe is the *Scheduler Universe*.
- Scheduler Universe jobs run on the submitting machine and serve as a meta-scheduler.
- DAGMan meta-scheduler included

DAGMan

- **Directed Acyclic Graph Manager**
- DAGMan allows you to specify the *dependencies* between your Condor jobs, so it can *manage* them automatically for you.
- (e.g., "Don't run job "B" until job "A" has completed successfully.")

What is a DAG?

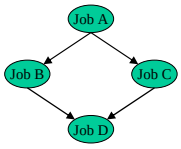
- A DAG is the data structure used by DAGMan to represent these dependencies.
- Each job is a "node" in the DAG.
- Each node can have any number of "parent" or "children" nodes – as lo as there are no loops!



Defining a DAG

- A DAG is defined by a *.dag file*, listing each of its nodes and their dependencies:

```
# diamond.dag
Job A a.sub
Job B b.sub
Job C c.sub
Job D d.sub
Parent A Child B C
Parent B C Child D
```



- each node will run the Condor job specified by its accompanying Condor submit file

Submitting a DAG

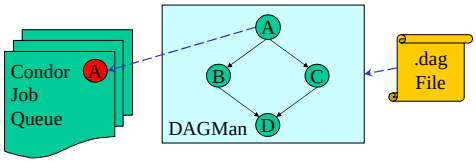
- To start your DAG, just run ***condor_submit_dag*** with your .dag file, and Condor will start a personal DAGMan daemon which to begin running your jobs:

```
% condor_submit_dag diamond.dag
```

- condor_submit_dag submits a Scheduler Universe Job with DAGMan as the executable.
- Thus the DAGMan daemon itself runs as a Condor job, so you don't have to baby-sit it.

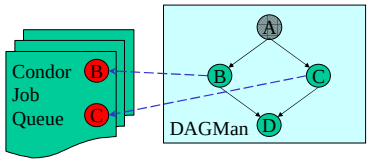
Running a DAG

- DAGMan acts as a "meta-scheduler", managing the submission of your jobs to Condor based on the DAG dependencies.



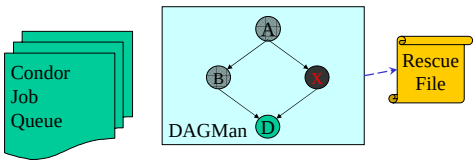
Running a DAG (cont'd)

- DAGMan holds & submits jobs to the Condor queue at the appropriate times.



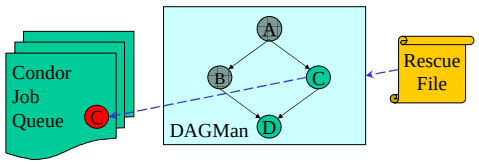
Running a DAG (cont'd)

- In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a *"rescue" file* with the current state of the DAG.



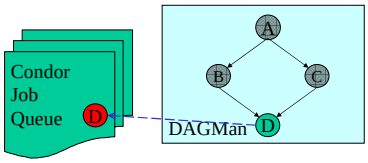
Recovering a DAG

- Once the failed job is ready to be re-run, the rescue file can be used to restore the prior state of the DAG.



Recovering a DAG (cont'd)

- Once that job completes, DAGMan will continue the DAG as if the failure never happened.



Finishing a DAG

- Once the DAG is complete, the DAGMan job itself is finished, and exits.