An Introduction to Using Condor

Condor Week 2012

Condor Project
Computer Sciences Department
University of Wisconsin-Madison

The Team - 2011

- established in 1985
- research and development of distributed high throughput computing

Today (May 1) is Miron's Birthday!

Condor is a High-Throughput Computing System

- Allows for many computational tasks to be completed over a long period of time
- Is concerned largely with the number of compute resources that are available to people who wish to use the system
- A very useful system for researchers and other users who are more concerned with the number of computations they can do over long spans of time, than they are with short-burst computations

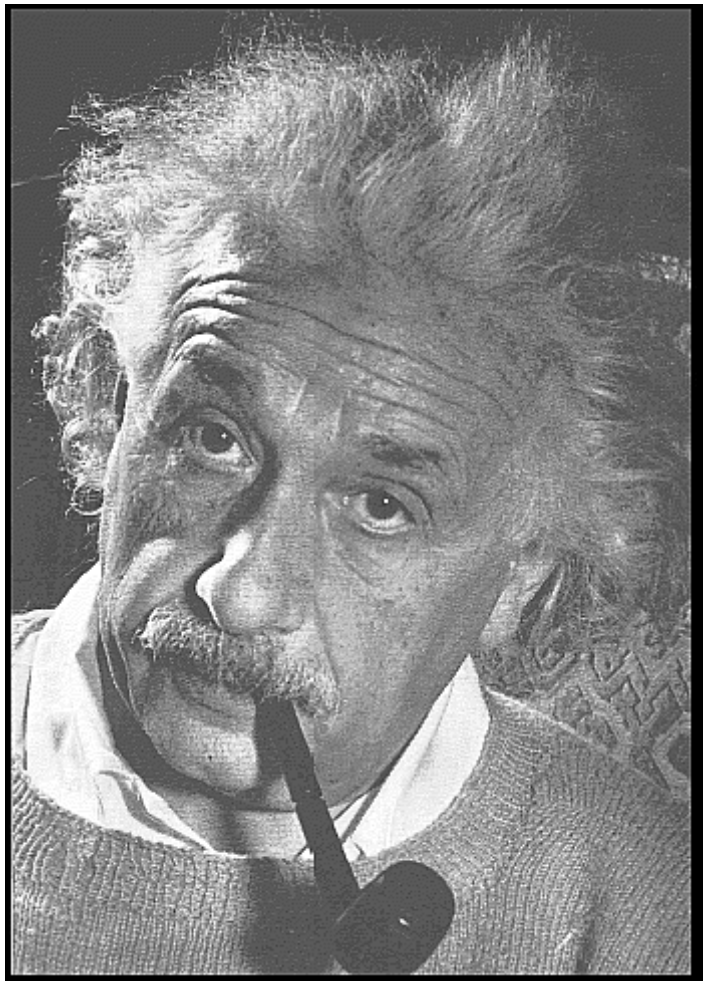
Condor's strengths

- Cycle scavenging works!
- High throughput computing
- Very configurable, adaptable
- Supports strong security methods
- Interoperates with many types of computing grids
- Facilities to manage both dedicated CPUs (clusters) and non-dedicated resources (desktops)
- Fault-tolerant: can survive crashes, network outages, any single point of failure.

Condor will ...

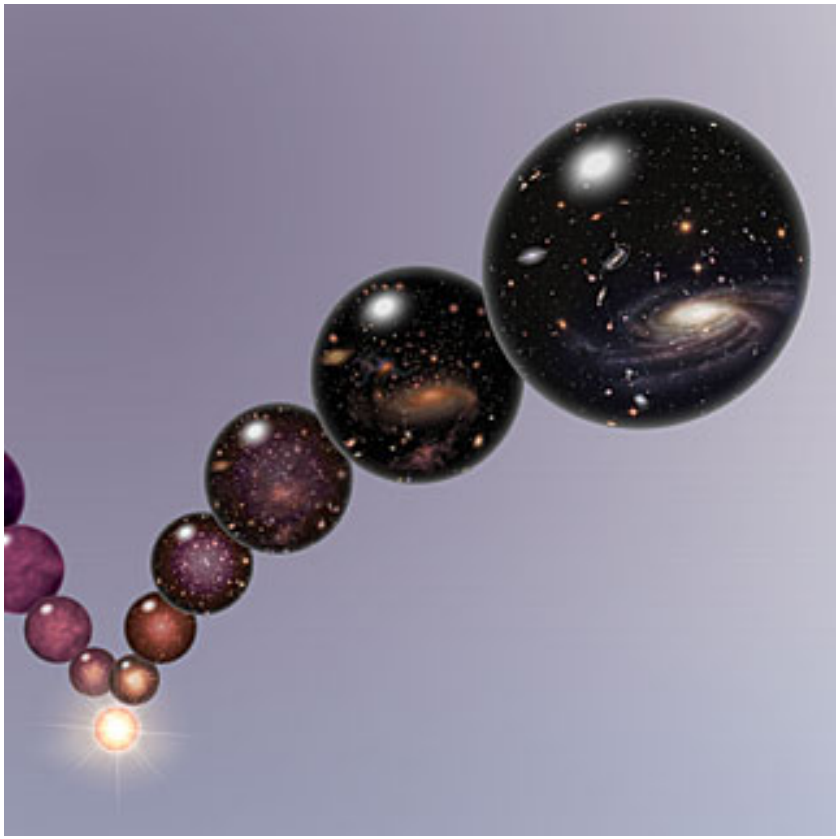
- Keep an eye on your jobs and will keep you posted on their progress
- Implement your policy on the execution order of the jobs
- Log your job's activities
- Add fault tolerance to your jobs
- Implement your policy as to when the jobs can run on your workstation

Our esteemed scientist*, has plenty of simulation to do.



* and Karen's cousin

Einstein's Simulation



Simulate the evolution of the cosmos, assuming various properties.

Simulation Overview

Varying values for each of:

- G (the gravitational constant): 100 values
- $\mathcal{R}_{\mu\nu}$ (the cosmological constant): 100 values
- c (the speed of light): 100 values

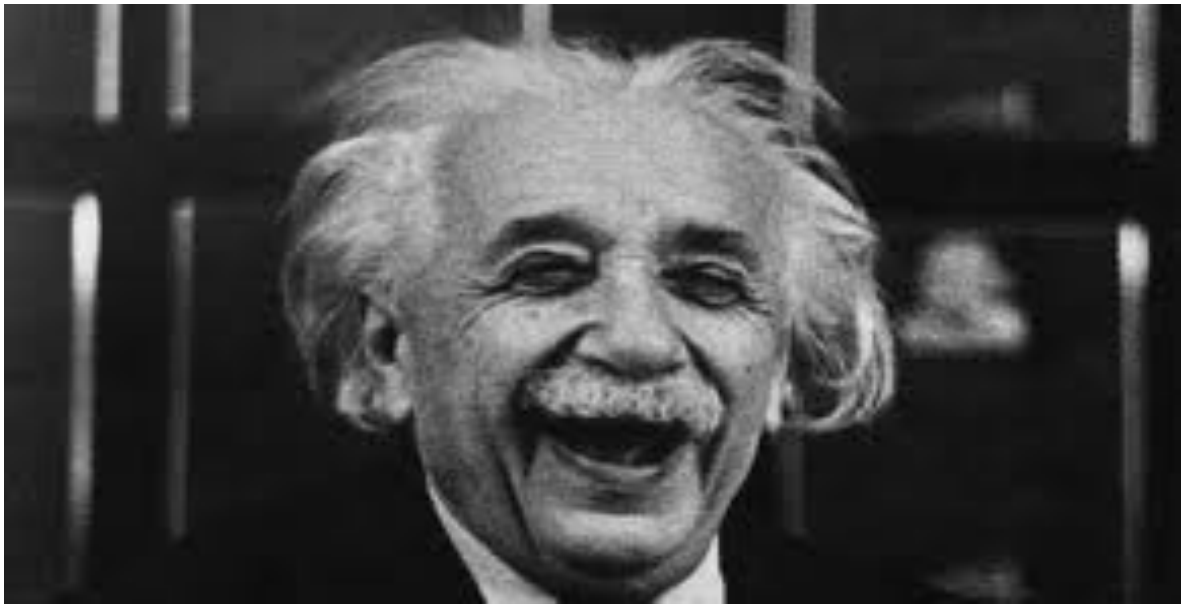
`100 × 100 × 100 = 1,000,000` jobs

Each job within the simulation:

- Requires up to 4 GBytes of RAM
- Requires 20 MBytes of input
- Requires 2 – 500 hours of computing time
- Produces up to 10 GBytes of output

Estimated total:

- 15,000,000 CPU hours or 1,700 compute YEARS
- 10 PetaBytes of output



Albert will be happy, since Condor will make the completion of this simulation easy.

Definitions

Job

- the Condor representation of a piece of work
- Condor's quanta of work
- Like a Unix process
- Can be an element of a workflow

ClassAd

- Condor's internal data representation

Machine or Resource

- computers that can do the processing

More Definitions

Match Making

- Associating a job with a machine resource

Central Manager

- Central repository for the whole pool
- Does match making

Submit Host

- The computer from which jobs are submitted to Condor

Execute Host

- The computer that runs a job

Jobs state their needs and preferences:

- Requirements (needs):
  - I **require** a Linux x86-64 platform
- Rank (preferences):
  - I **prefer** the machine with the most memory
  - I **prefer** a machine in the botany department

Machines also specify needs and preferences:

- Requirements (needs):
  - **Require** that jobs run only when there is no keyboard activity
  - **Never** run jobs belonging to Dr. Heisenberg
- Rank (preferences):
  - I **prefer** to run Albert's jobs

Condor ClassAds

the language that Condor uses to represent information – about jobs (job ClassAd), machines (machine ClassAd), and programs that implement Condor's functionality (called daemons), etc.

ClassAd Structure

semi-structured
user-extensible
schema-free

AttributeName = Value
or
AttributeName = Expression

Part of a Job ClassAd

```
MyType          = "Job"
TargetType      = "Machine"
ClusterId       = 1
ProcId          = 0
IsPhysics       = True
Owner           = "einstein"
Cmd             = "cosmos"
Requirements    = (Arch == "INTEL")
.
.
.
```

String

Integer

Boolean

Boolean Expression

The Magic of Matchmaking

The Condor match maker matches job ClassAds with machine ClassAds, taking into account:

- Requirements of both the machine *and* the job
- Rank of both the job *and* the machine
- Priorities, such as those of users and also group priorities

Getting Started:

1. Choose a universe for the job
2. Make the job batch-ready
   - includes making the input data available and accessible
3. Create a submit description file
4. Run `condor_submit` to put the job(s) in the queue

1. Choose the Universe

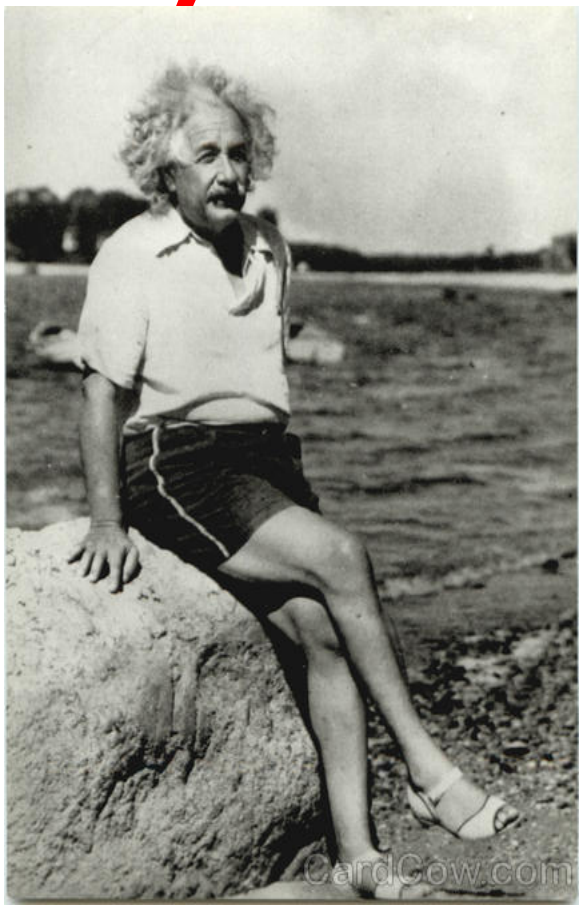
- controls how Condor handles jobs
- Condor's many universes include:
  - vanilla
  - standard
  - grid
  - java
  - parallel
  - vm

Using the Vanilla Universe

- Allows running almost any "serial" job
- Provides automatic file transfer for input and output files
- Like vanilla ice cream, can be used in just about any situation

2. Make the job batch-ready

- Must be able to run in the background
- No interactive input
- No GUI/window clicks



Batch-Ready: Standard Input & Output

- Job can still use **STDIN**, **STDOUT**, and **STDERR** (the keyboard and the screen), but files are used for these instead of the actual devices
- Similar to Unix shell:

```
$ ./myprogram <input.txt >output.txt
```

Make the Data Available

- Condor will
  - Transfer data files *to* the job
  - Transfer results files *back from* the job
- Place the job's data files in a place where Condor can access them

3. Create a Submit Description File

- A plain ASCII text file
- File name extensions are irrelevant
  - Many use `.sub` or `.submit` as suffixes
- Tells Condor about the job
- Can describe many jobs at once (a cluster), each with different input, output, command line arguments, etc.

Simple Submit Description File

```
# file name is cosmos.sub
# (Lines beginning with # are comments)
# NOTE: the commands on the left are not
#   case sensitive, but file names
#   (on the right) are!

Universe   = vanilla
Executable = cosmos
Input      = cosmos.in
Output     = cosmos.out
Log        = cosmos.log
Queue
```

Put 1 instance of the job in the queue

Input, Output, and Error Files

- Read job's standard input from `in_file`:

  `Input = in_file`

  like shell: `$ program < in_file`

- Write job's standard output to `out_file`:

  `Output = out_file`

  like shell: `$ program > out_file`

- Write job's standard error to `error_file`:

  `Error = error_file`

  like shell: `$ program 2> error_file`

Logging the Job's Activities

- In the submit description file:

```
log = cosmos.log
```

- Creates a log of job events, which is

  The Life Story of a Job

  - Shows all events in the life of a job
- Good advice: *always* have a log file

Sample Portion of Job Log

```
000 (0101.000.000) 05/25 19:10:03 Job submitted from host:
<128.105.146.14:1816>
...
001 (0101.000.000) 05/25 19:12:17 Job executing on host:
<128.105.146.14:1026>
...
005 (0101.000.000) 05/25 19:13:06 Job terminated.
    (1) Normal termination (return value 0)
...
```

000, 001, and 005 are examples of event numbers.

4. Submit the Job

- Run `condor_submit`, providing the name of the submit description file:

```
$ condor_submit cosmos.sub
Submitting job(s).
1 job(s) submitted to cluster 100.
```

- `condor_submit` then
  - parses the submit description file, checking for errors
  - creates a ClassAd that describes the job(s)
  - places the job in the queue
  - an atomic operation, with two-phase commit

Observe Jobs in the Queue

```
$ condor_q
-- Submitter: submit.chtc.wisc.edu : <128.104.55.9:51883> :
   submit.chtc.wisc.edu
ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
2.0      heisenberg    1/13 13:59    0+00:00:00 R  0    0.0  env
3.0      hawking       1/15 19:18    0+04:29:33 H  0    0.0  script.sh
4.0      hawking       1/15 19:33    0+00:00:00 H  0    0.0  script.sh
5.0      hawking       1/15 19:33    0+00:00:00 H  0    0.0  script.sh
6.0      hawking       1/15 19:34    0+00:00:00 H  0    0.0  script.sh
...
96.0     bohr          4/5  13:46    0+00:00:00 I  0    0.0  atoms H
97.0     bohr          4/5  13:46    0+00:00:00 I  0    0.0  atoms H
98.0     bohr          4/5  13:52    0+00:00:00 I  0    0.0  atoms H
99.0     bohr          4/5  13:52    0+00:00:00 I  0    0.0  atoms H
100.0    einstein      4/5  13:55    0+00:00:00 I  0    0.0  cosmos

100 jobs; 1 completed, 0 removed, 20 idle, 1 running, 77 held,
   0 suspended
```

File Transfer

Beyond `STDIN`, `STDOUT`, and `STDERR`, Condor can transfer other files

- `Transfer_Input_Files` specifies a list of files for Condor to transfer from the submit machine to the execute machine
- `Transfer_Output_Files` specifies a list of files for Condor to transfer back from the execute machine to the submit machine
- If `Transfer_Output_Files` is *not* specified, Condor will transfer back all "new" files in the execute directory

Transferring Files

Files need to get from the submit machine to the execute machine. 2 possibilities:

1. both machines have access to a shared file system
2. machines are have separate file systems

Should_Transfer_Files

- `YES`: Transfer files to execution machine
- `NO`: Rely on shared file system
- `IF_NEEDED`: Automatically transfer the files, if the submit and execute machine are not in the same `FileSystemDomain` (Translation: Use shared file system if available)

When_To_Transfer_Output

- `ON_EXIT`: Transfer output files only when job completes
- `ON_EXIT_OR_EVICT`: Transfer output files when job completes or is evicted

File Transfer Example

```
# new cosmos.sub file
Universe                        = vanilla
Executable                       = cosmos
Log                             = cosmos.log
Transfer_Input_Files             = cosmos.dat
Transfer_Output_Files           = results.dat
Should_Transfer_Files           = IF_NEEDED
When_To_Transfer_Output         = ON_EXIT
Queue
```

Command Line Arguments

```
# Example with command line arguments
Universe   = vanilla
Executable = cosmos
Arguments  = -c 299792458 -G 6.67300e-112

. . .

Queue
```

Invokes executable with

```
cosmos -c 299792458 -G 6.673e-112
```

Look at the `condor_submit` man page to see formatting for `Arguments`. This example has argc = 5.

More Feedback

- Condor sends email about job events to the submitting user
- Specify *one* of these in the submit description file:

```
Notification = complete
Notification = never
Notification = error
Notification = always
```

Default

ClusterId.ProcID is Job ID

- If the submit description file describes multiple jobs, it is called a cluster
- Each cluster has a cluster number, where the cluster number is unique to the job queue on a machine
- Each individual job within a cluster is called a process, and process numbers always start at zero
- A Condor Job ID is the cluster number, a period, and the process number
  - Job ID = 20.0 — **Cluster 20, process 0**
  - Job IDs: 21.0, 21.1, 21.2 — **Cluster 21, process 0, 1, 2**

1 Cluster

```
Universe   = vanilla
Executable = cosmos

log        = cosmos_0.log
Input      = cosmos_0.in
Output     = cosmos_0.out

Queue                    Job 102.0 (cluster 102, process 0)

log        = cosmos_1.log
Input      = cosmos_1.in
Output     = cosmos_1.out

Queue                    Job 102.1 (cluster 102, process 1)
```

File Organization

A logistical nightmare places all input, output, error and log files in one directory

- 3 files × 1,000,000 jobs = 3,000,000 files
- The submit description file is 4,000,000+ lines

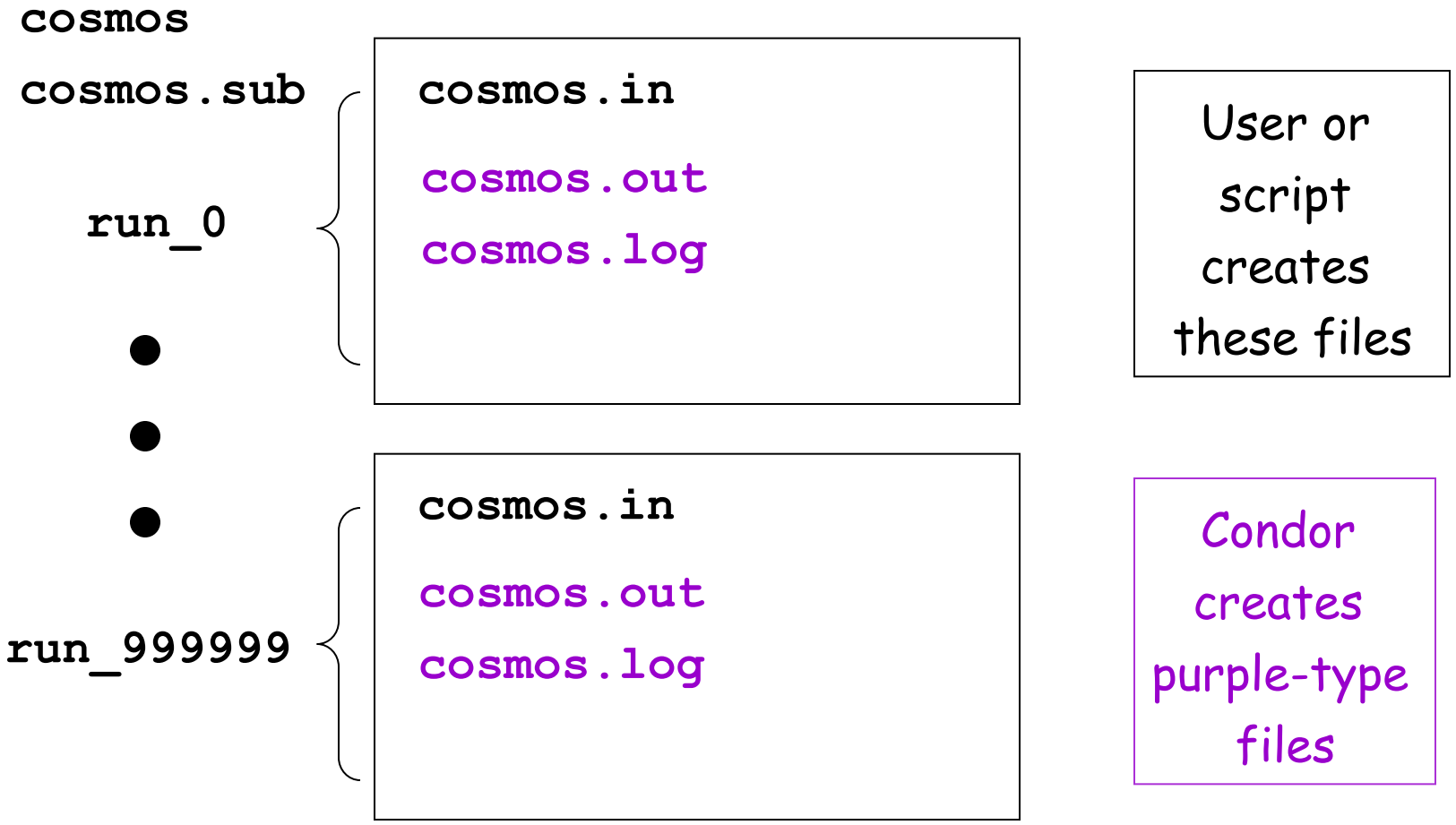
The directory will be difficult (at best) to sort through

Better Organization

- Create subdirectories for each run, specifically named
  - `run_0`, `run_1`, … `run_999999`
- Implement creation of directories with a Python or Perl program
- Create input files in each of these
  - `run_0/cosmos.in`
  - `run_1/cosmos.in`
  - …
  - `run_999999/cosmos.in`

- The output, error & log files for each job will be created by Condor when the job runs

Einstein's simulation directory



Submit Description File

```
# Cluster of 1,000,000 jobs with
# different directories
Universe                = vanilla
Executable              = cosmos
Log                     = cosmos.log
Output                  = cosmos.out
Input                   = cosmos.in

...

InitialDir = run_0
Queue                                Job 103.0 (Cluster 103, Process 0)

InitialDir = run_1
Queue                                Job 103.1 (Cluster 103, Process 1)
```

This file contains 999,998 more instances of `InitialDir` and `Queue`.

An Even Better Way

- Queue all 1,000,000 processes with a single command:

```
Queue 1000000
```

- Within the submit description file, Condor provides macros

  `$(Process)` will be expanded to the process number for each job in the cluster

  0 - 999999 for the 1,000,000 jobs

Using `$(Process)`

- The initial directory for each job can be specified using `$(Process)`

```
InitialDir = run_$(Process)
```

  - Condor will expand these directories to

    `run_0`, `run_1`, … `run_999999`

- Similarly, arguments could use a macro to pass a unique ID to each job instance

```
Arguments = -n $(Process)
```

  - Condor will expand these to:

```
-n 0
-n 1
…
-n 999999
```

(Best) Submit Description File

```
# Example defining a cluster of
# 1,000,000 jobs
Universe   =  vanilla
Executable = cosmos
Log        = cosmos.log
Input      = cosmos.in
Output     = cosmos.out
InitialDir = run_$(Process)
Queue 1000000
```

Finally, Albert submits this. Be patient, it'll take a while…

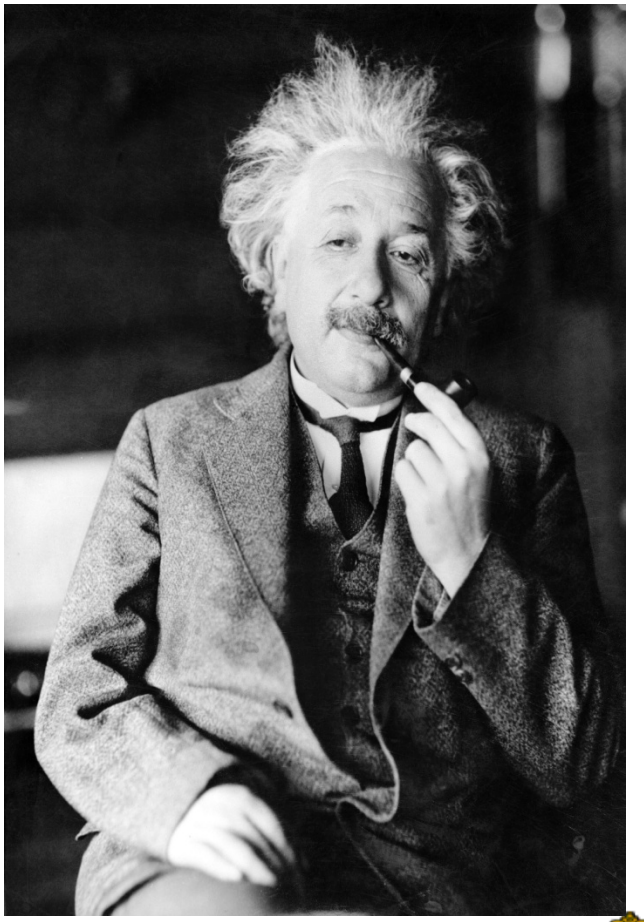
```
$ condor_submit cosmos.sub
Submitting
   job(s) ..........................................
   ..........................................
   ..........................................
   ..........................................
   ..........................................
   ..........
Logging submit
   event(s) ........................................
   ..........................................
   ..........................................
   ..........................................
   ..........................................
   ..........
1000000 job(s) submitted to cluster 104.
```

The Job Queue

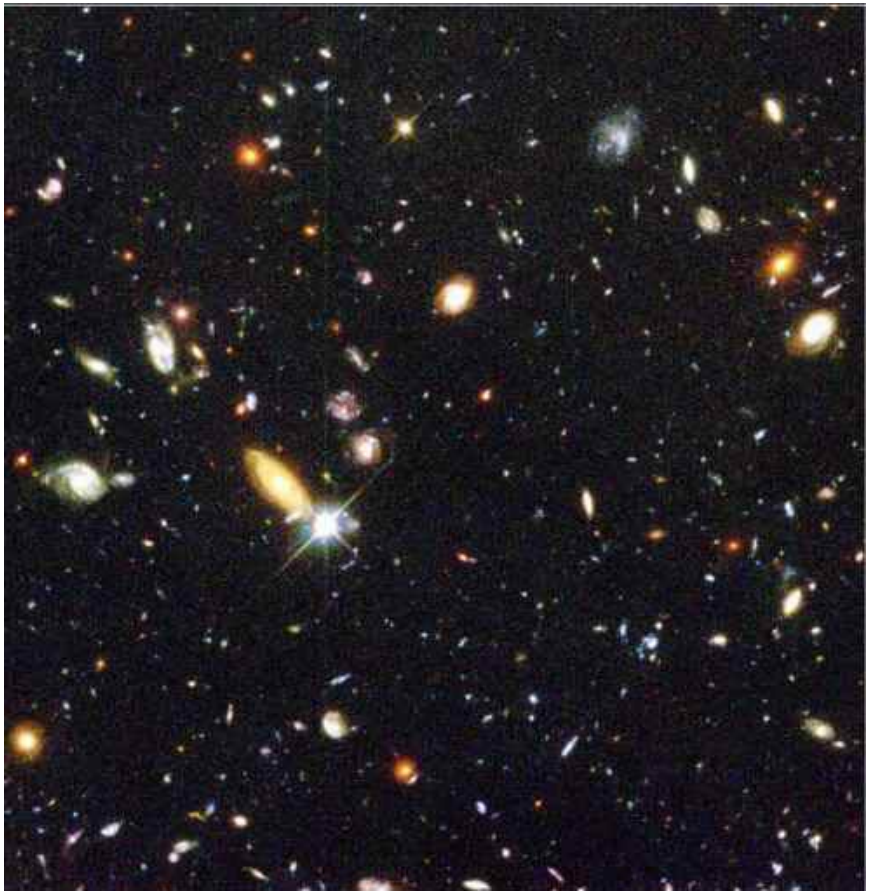
```
$ condor_q
-- Submitter: submit.chtc.wisc.edu :
  <128.104.55.9:51883> : submit.chtc.wisc.edu
ID        OWNER   SUBMITTED   RUN_TIME  ST PRI SIZE CMD
104.0       einstein 4/20 12:08 0+00:00:05 R 0 9.8 cosmos
104.1       einstein 4/20 12:08 0+00:00:03 I 0 9.8 cosmos
104.2       einstein 4/20 12:08 0+00:00:01 I 0 9.8 cosmos
104.3       einstein 4/20 12:08 0+00:00:00 I 0 9.8 cosmos
...
104.999998 einstein 4/20 12:08 0+00:00:00 I 0 9.8 cosmos
104.999999 einstein 4/20 12:08 0+00:00:00 I 0 9.8 cosmos

999999 jobs; 999998 idle, 1 running, 0 held
```

Albert Relaxes

- Condor watches over the jobs, and will restart them if required, etc.
- Time for a cold one!



More That Condor Can Do



Remove Jobs with condor_rm

- You can only remove jobs that you own
- Privileged user can remove any jobs
  - "root" on Linux
  - "administrator" on Windows

`condor_rm 4` Removes all cluster 4 jobs

`condor_rm 4.2` Removes only the job with job ID 4.2

`condor_rm -a` Removes all of your jobs. *Careful !*

Specify Job Requirements

- A boolean expression (syntax similar to C or Java)
- Evaluated with attributes from machine ClassAd(s)
- **Must** evaluate to True for a match to be made

```
Universe     = vanilla
Executable      = mathematica

...

Requirements = (  \
   HasMathematicaInstalled =?= True )
Queue 20
```

Specify Needed Resources

New in 7.7.6

Items appended to job **Requirements**

- **request_memory** – the amount of memory (in Mbytes) that the job needs to avoid excessive swapping
- **request_disk** – the amount of disk space (in Kbytes) that the job needs. Will be sum of space for executable, input files, output files and temporary files. Default is size of initial sandbox (executable plus input files).
- **request_cpus** – the number of CPUs (cores) that the job needs. Defaults to 1.

Specify Job Rank

- All matches which meet the requirements can be sorted by preference with a **Rank** expression
  - Numerical
  - Higher rank values match first
- Like **Requirements**, is evaluated against attributes from machine ClassAds

```
Universe     = vanilla
Executable     = cosmos

. . .

Rank           = (KFLOPS*10000) + Memory
Queue 1000000
```

Job Policy Expressions

- Do not remove if exits with a signal:

```
on_exit_remove = ExitBySignal == False
```

- Place on hold if exits with nonzero status or ran for less than an hour:

```
on_exit_hold =
  ( (ExitBySignal==False) && (ExitSignal != 0) ) ||
  ( (ServerStartTime - JobStartDate) < 3600)
```

- Place on hold if job has spent more than 50% of its time suspended:

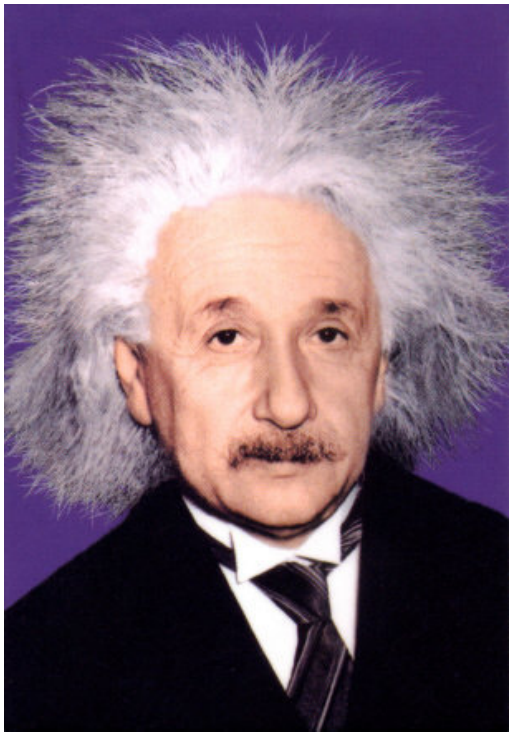
```
periodic_hold =
  ( CumulativeSuspensionTime >
    (RemoteWallClockTime / 2.0) )
```

Running lots of Short-Running Jobs

- Know that starting a job in Condor is somewhat expensive, in terms of time
- 3 items that might help:

1. Batch your short jobs together
   - Write a wrapper script that will run a set of the jobs in series
   - Submit the wrapper script as your job
2. Explore Condor's parallel universe
3. There are some configuration parameters that may be able to help
   - Contact a Condor staff person for more info

Common Problems with Jobs



Jobs Are Idle

Our scientist runs `condor_q` and finds all his jobs are idle

```
$ condor_q
-- Submitter: x.cs.wisc.edu : <128.105.121.53:510>
   :x.cs.wisc.edu
ID   OWNER      SUBMITTED     RUN_TIME   ST PRI SIZE CMD
5.0 einstein   4/20 12:23   0+00:00:00   I 0    9.8   cosmos
5.1 einstein   4/20 12:23   0+00:00:00   I 0    9.8   cosmos
5.2 einstein   4/20 12:23   0+00:00:00   I 0    9.8   cosmos
5.3 einstein   4/20 12:23   0+00:00:00   I 0    9.8   cosmos
5.4 einstein   4/20 12:23   0+00:00:00   I 0    9.8   cosmos
5.5 einstein   4/20 12:23   0+00:00:00   I 0    9.8   cosmos
5.6 einstein   4/20 12:23   0+00:00:00   I 0    9.8   cosmos
5.7 einstein   4/20 12:23   0+00:00:00   I 0    9.8   cosmos
8 jobs; 8 idle, 0 running, 0 held
```

Exercise a little patience

- On a busy pool, it can take a while to match jobs to machines, and then start the jobs
- Wait at least a negotiation cycle or two, typically a few minutes

Look in the Job Log

It will likely contain clues:

```
$ cat cosmos.log
000 (031.000.000) 04/20 14:47:31 Job submitted from
  host: <128.105.121.53:48740>
...
007 (031.000.000) 04/20 15:02:00 Shadow exception!
        Error from starter on gig06.stat.wisc.edu:
  Failed to open '/scratch.1/einstein/workspace/v76/
  condor-test/test3/run_0/cosmos.in' as standard
  input: No such file or directory (errno 2)
        0  -  Run Bytes Sent By Job
        0  -  Run Bytes Received By Job
...
```

Check Machine's Status

```
$ condor_status
Name                OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime
slot1@c002.chtc.wi  LINUX      X86_64 Claimed   Busy     1.000  4599  0+00:10:13
slot2@c002.chtc.wi  LINUX      X86_64 Claimed   Busy     1.000  1024  1+19:10:36
slot3@c002.chtc.wi  LINUX      X86_64 Claimed   Busy     0.990  1024  1+22:42:20
slot4@c002.chtc.wi  LINUX      X86_64 Claimed   Busy     1.000  1024  0+03:22:10
slot5@c002.chtc.wi  LINUX      X86_64 Claimed   Busy     1.000  1024  0+03:17:00
slot6@c002.chtc.wi  LINUX      X86_64 Claimed   Busy     1.000  1024  0+03:09:14
slot7@c002.chtc.wi  LINUX      X86_64 Claimed   Busy     1.000  1024  0+19:13:49
...
vm1@INFOLABS-SML65  WINDOWS    INTEL  Owner     Idle     0.000   511  [Unknown]
vm2@INFOLABS-SML65  WINDOWS    INTEL  Owner     Idle     0.030   511  [Unknown]
vm1@INFOLABS-SML66  WINDOWS    INTEL  Unclaimed Idle     0.000   511  [Unknown]
vm2@INFOLABS-SML66  WINDOWS    INTEL  Unclaimed Idle     0.010   511  [Unknown]
vm1@infolabs-smlde  WINDOWS    INTEL  Claimed   Busy     1.130   511  [Unknown]
vm2@infolabs-smlde  WINDOWS    INTEL  Claimed   Busy     1.090   511  [Unknown]

                     Total Owner Claimed Unclaimed Matched Preempting Backfill

       INTEL/WINDOWS   104    78      16        10       0          0        0
        X86_64/LINUX   759   170     587         0       0          1        0

               Total   863   248     603        10       0          1        0
```

Never matched?
condor_q -analyze

```
$ condor_q -ana 29
The Requirements expression for your job is:

( (target.Memory > 8192) ) && (target.Arch == "INTEL") &&
(target.OpSys == "LINUX") && (target.Disk >= DiskUsage) &&
(TARGET.FileSystemDomain == MY.FileSystemDomain)
Condition                Machines    Matched  Suggestion
---------                ----------- -------- -----------
1 ( ( target.Memory > 8192 ) )  0          MODIFY TO 4000
2 ( TARGET.FileSystemDomain == "cs.wisc.edu" )584
3   ( target.Arch == "INTEL" )  1078
4   ( target.OpSys == "LINUX" ) 1100
5   ( target.Disk >= 13 )       1243
```

Learn about available resources:

```
$ condor_status -const 'Memory > 8192'
(no output means no matches)

$ condor_status -const 'Memory > 4096'
Name               OpSys   Arch   State     Activ LoadAv Mem   ActvtyTime
vm1@s0-03.cs.  LINUX   X86_64 Unclaimed Idle  0.000  5980  1+05:35:05
vm2@s0-03.cs.  LINUX   X86_64 Unclaimed Idle  0.000  5980 13+05:37:03
vm1@s0-04.cs.  LINUX   X86_64 Unclaimed Idle  0.000  7988  1+06:00:05
vm2@s0-04.cs.  LINUX   X86_64 Unclaimed Idle  0.000  7988 13+06:03:47

                     Total Owner Claimed Unclaimed Matched Preempting
   X86_64/LINUX          4     0       0         4       0          0
          Total          4     0       0         4       0          0
```

Interact With A Job

- Perhaps a job is running for much longer than expected.
  - Is it stuck accessing a file?
  - Is it in an infinite loop?
- Try `condor_ssh_to_job`
  - Interactive debugging in Unix
  - Use *ps*, *top*, *gdb*, *strace*, *lsof*, …
  - Forward ports, X, transfer files, etc.
  - Currently not available on Windows

Interactive Debug Example

```
$ condor_q
-- Submitter: cosmos.phy.wisc.edu : <128.105.165.34:1027>

ID     OWNER     SUBMITTED   RUN_TIME   ST PRI SIZE CMD
1.0   einstein 4/15 06:52 1+12:10:05 R 0    10.0 cosmos

1 jobs; 0 idle, 1 running, 0 held

$ condor_ssh_to_job 1.0

Welcome to slot4@c025.chtc.wisc.edu!
Your condor job is running with pid(s) 15603.

$ gdb -p 15603
. . .
```

Condor is extremely flexible. Here are overviews of some of the many features that you may want to learn more about.

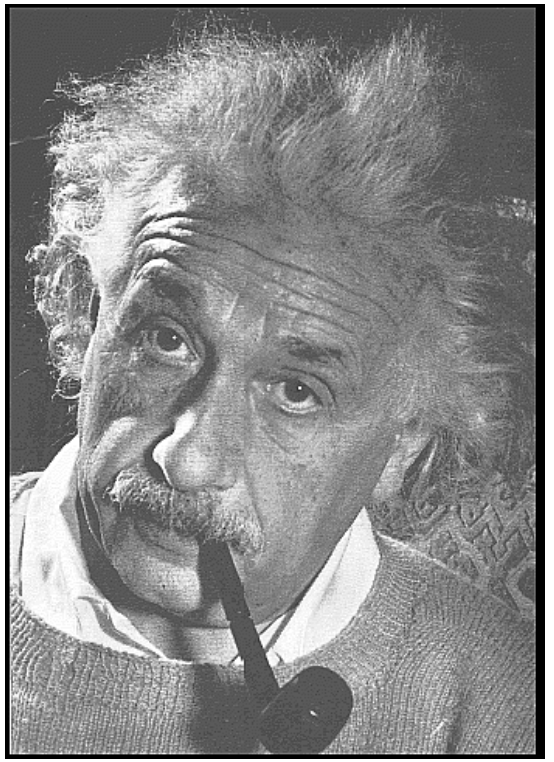
After this tutorial, here are some places you might find help:

1. Condor manual
2. condor-users mailing list. See

`https://lists.cs.wisc.edu/mailman/listinfo/condor-users`

3. Wiki
   See https://condor-wiki.cswisc.edu/index.cgi/wiki
4. Developers

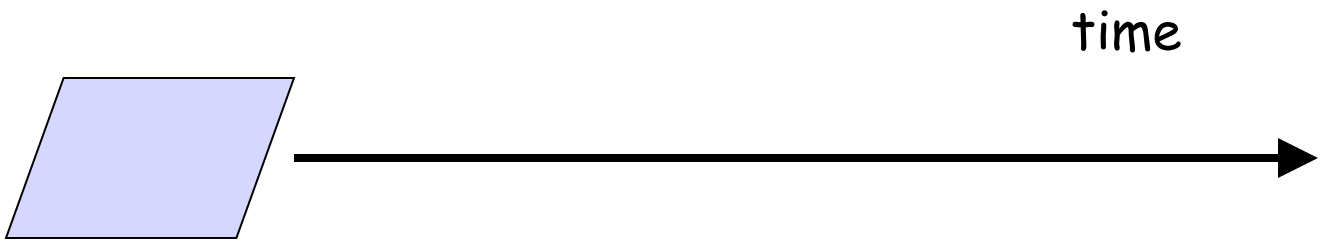
- The more time a job takes to run, the higher the risk of
  - being preempted by a higher priority user or job
  - getting kicked off a machine (vacated), because the machine has something else it prefers to do
- Condor's standard universe may provide a solution.



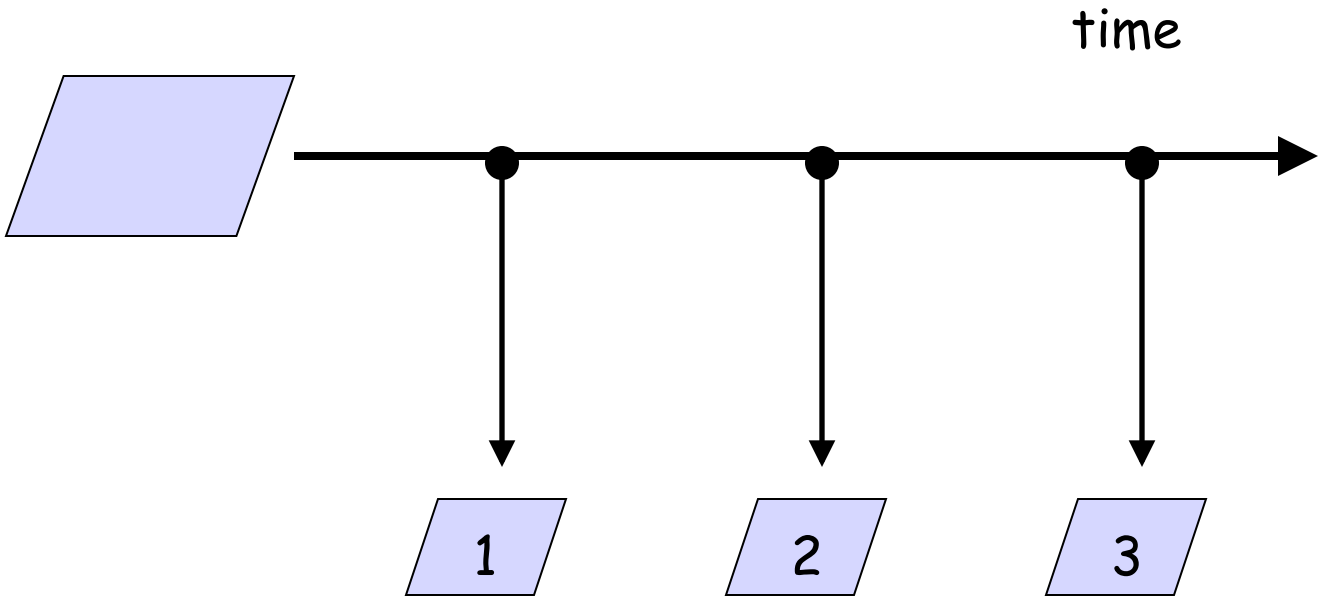
Standard Universe

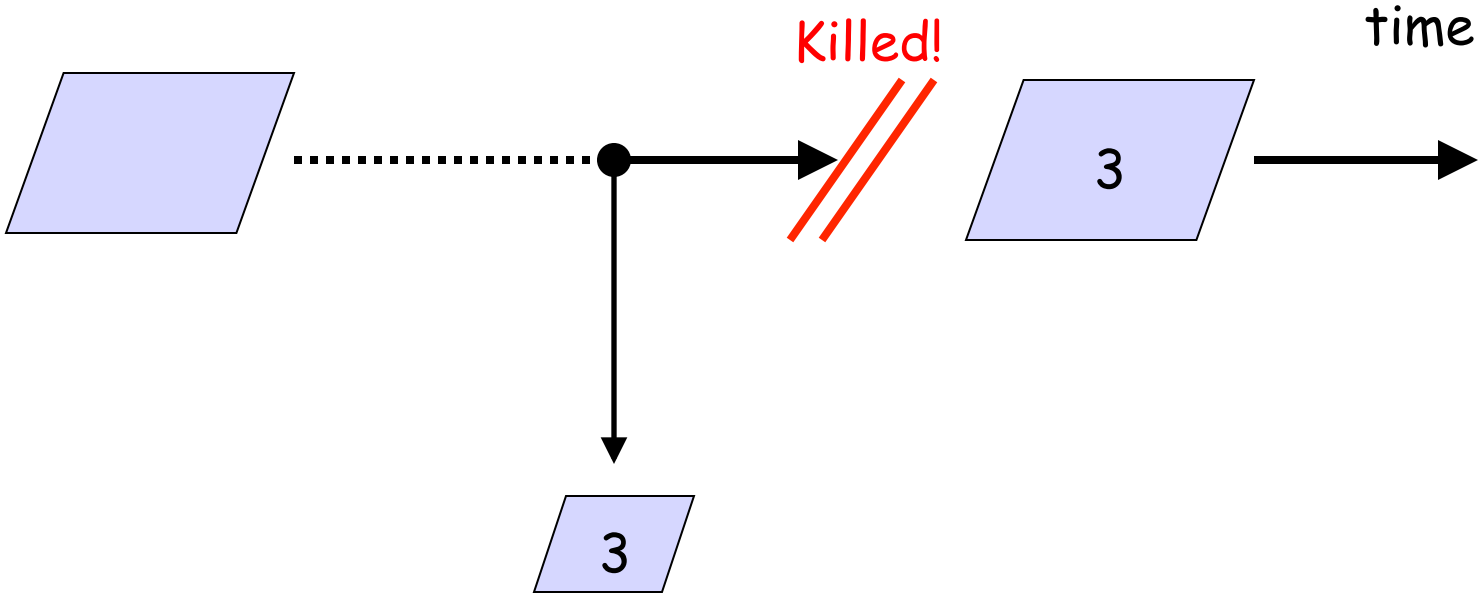
- Regularly while the job runs, or when the job is to be kicked off the machine, Condor takes a checkpoint -- a complete state of the job.
- With a checkpoint, the job can be matched to another machine, and *continue on*.

checkpoint: the entire state of a program, saved in a file, such as CPU registers, memory image, I/O, etc.

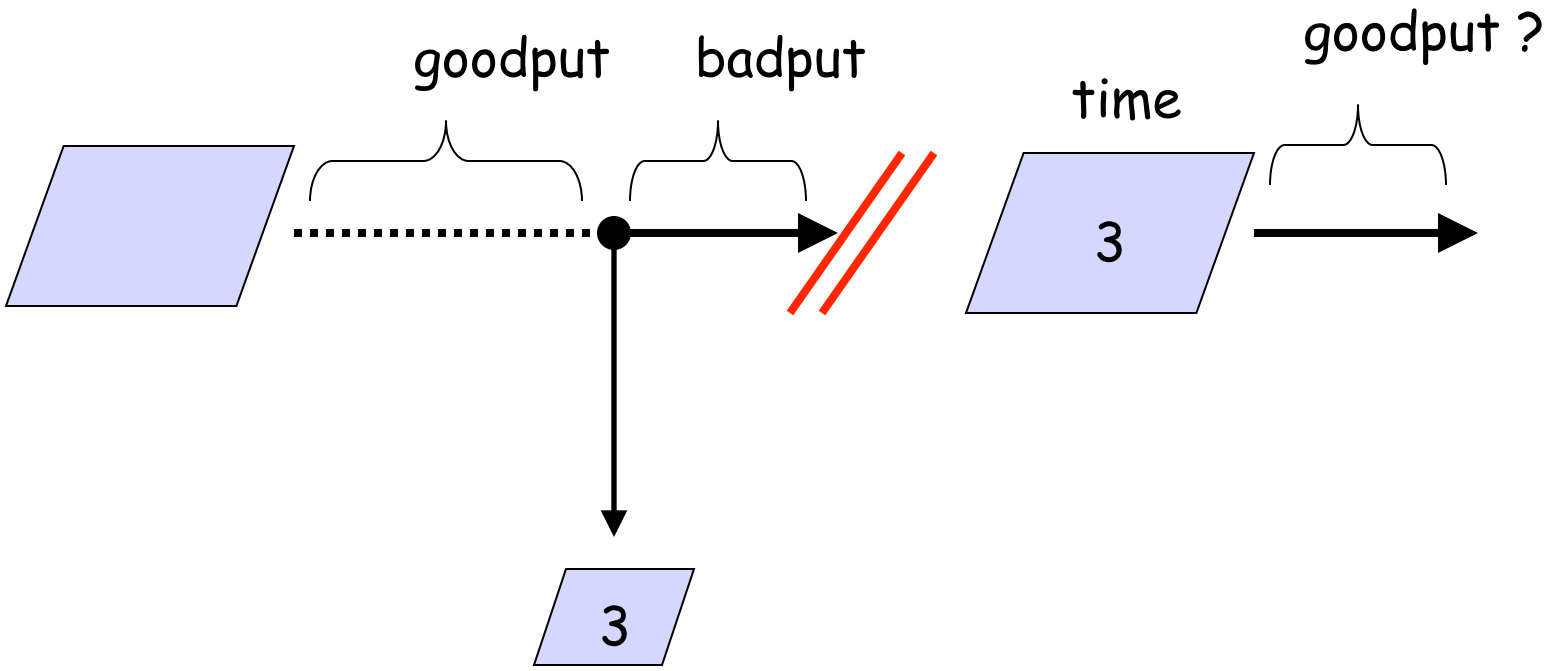


3 Checkpoints




Killed!
time
3
3

Goodput and Badput



Standard Universe Features

- Remote system calls (remote I/O)
  - The job can read / write files as if they were local
- No source code changes typically required, but relinking the executable with Condor's standard universe support library is required.
- Programming language independent

How to Relink

Place **condor_compile** in front of the command used to link the job:

```
$ condor_compile gcc -o myjob myjob.c
```

- OR -

```
$ condor_compile f77 -o myjob filea.f fileb.f
```

- OR -

```
$ condor_compile make -f MyMakefile
```

Limitations

- Condor's checkpoint mechanism is not at the kernel level. Therefore, a standard universe job may *not*:
  - `fork()`
  - Use kernel threads
  - Use some forms of IPC, such as pipes and shared memory
- Must have access to object code in order to relink
- Only available on some Linux platforms

Parallel Universe

- When multiple processes must be running at the same time on different machines.
- Provides a mechanism for controlling parallel algorithms
  - Fault tolerant
  - Allows for resources to come and go
  - Ideal for Computational Grid settings
- Especially for MPI

MPI Job Submit Description File

```
# MPI job submit description file
universe = parallel
executable = mp1script
arguments = my_mpich_linked_exe arg1 arg2
machine_count = 4
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = my_mpich_linked_exe
queue
```

MPI jobs

Note: Condor will probably not schedule all of the jobs on the same machine, so consider using whole machine slots

See the Condor Wiki:
Under *How To Admin Recipes*,
"How to allow some jobs to claim the whole machine instead of one slot"

VM Universe

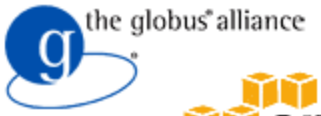
- A virtual machine instance is the Condor job
- The vm universe offers
  - Job sandboxing
  - Checkpoint and migration
  - Safe elevation of privileges
  - Cross-platform submission
- Condor supports VMware, Xen, and KVM
- Input files can be imported as CD-ROM image
- When the VM shuts down, the modified disk image is returned as job output

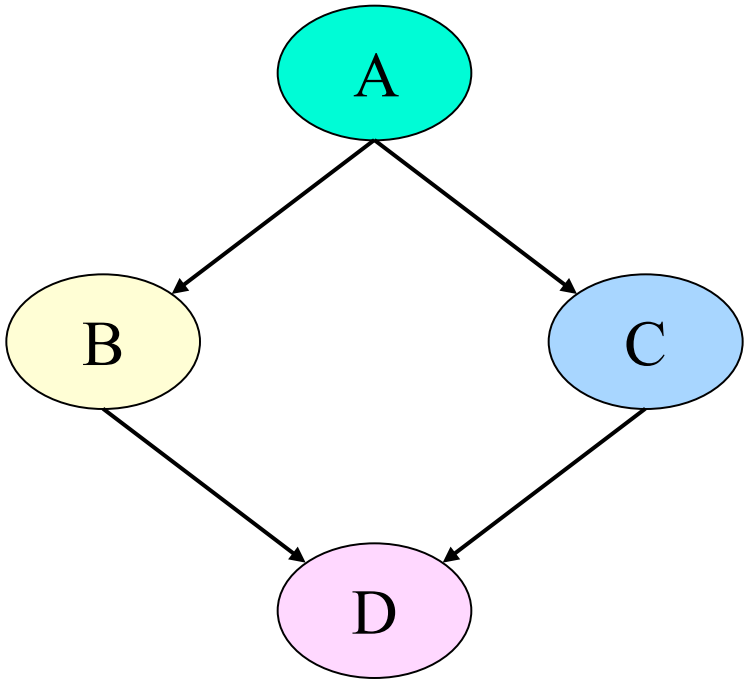
Machine Resources are Numerous: The Grid

Given access (authorization) to grid resources , as well as certificates (for authentication) and access to Globus or other resources at remote institutions, Condor's grid universe does the trick !

Grid Universe

- All specification is in the submit description file
- Supports many "back end" types:
  - Globus: GT2, GT5
  - NorduGrid
  - UNICORE
  - Condor
  - PBS
  - LSF
  - SGE
  - EC2
  - Deltacloud
  - Cream





- Some sets of jobs have dependencies.
- Condor handles them with DAGMan.
- See Nathan's tutorial. Today at 11:30am.

the Java Universe

- Java Universe provides more than just inserting "java" at the start of the execute line of a vanilla job:
  - Knows which machines have a JVM installed
  - Knows the location, version, and performance of JVM on each machine
  - Knows about jar files, etc.
  - Provides more information about Java job completion than just JVM exit code
    - Program runs in a Java wrapper, allowing Condor to report Java exceptions, etc.

Java Universe Example

```
# Example Java Universe Submit file
Universe   = java
Executable = Main.class
jar_files  = MyLibrary.jar
Input      = infile
Output     = outfile
Arguments  = Main 1 2 3
Queue
```

In Review

With Condor's help, both you and Albert can:

- Submit jobs
- Manage jobs
- Organize data files
- Identify aspects of universe choice

Thank you!

Check us out on the web:

http://www.condorproject.org

Email:

condor-admin@cs.wisc.edu

Extra Slides with More Information You Might Want to Reference

InitialDir

- Identifies a directory for file input and output.
- Also provides a directory (on the submit machine) for the user log, when a full path is not specified.
- **Note:** `Executable` is **not** relative to `InitialDir`

```
# Example with InitialDir
Universe   = vanilla
InitialDir = /home/einstein/cosmos/run
Executable = cosmos
Log        = cosmos.log
Input      = cosmos.in
Output     = cosmos.out
Error      = cosmos.err

Transfer_Input_Files=cosmos.dat
Arguments  = -f cosmos.dat
Queue
```

NOT Relative to `InitialDir`

Is Relative to `InitialDir`

Substitution Macro

`$$(<attribute>)` will be replaced by the value of the specified attribute from the Machine ClassAd

Example:

Machine ClassAd has:

```
CosmosData = "/local/cosmos/data"
```

Submit description file has

```
Executable   = cosmos
Requirements = (CosmosData =!= UNDEFINED)
Arguments    = -d $$(CosmosData)
```

Results in the job invocation:

```
cosmos -d /local/cosmos/data
```

Getting Condor

- Available as a free download from
  http://www.cs.wisc.edu/condor
- Download Condor for your operating system
  - Available for most modern UNIX platforms (including Linux and Apple's OS/X)
  - Also for Windows XP / Vista / Windows 7
- Repositories
  - YUM: RHEL 4 & 5
    - `$ yum install condor`
  - APT: Debian 4 & 5
    - `$ apt-get install condor`

Condor Releases

- Stable / Developer Releases
  - Version numbering scheme similar to that of the (pre 2.6) Linux kernels …
- Major.minor.release
  - If minor is even (a.b.c): Stable series
    - Very stable, mostly bug fixes
    - Current: 7.6
    - Examples: 7.4.5, 7.6.0
      - 7.6.0 just released
  - If minor is odd (a.b.c): Developer series
    - New features, may have some bugs
    - Current: 7.7
    - Examples: 7.5.2, 7.7.0
      - 7.7.0 in the works

General User Commands

condor_status	View Pool Status
condor_q	View Job Queue
condor_submit	Submit new Jobs
condor_rm	Remove Jobs
condor_prio	Intra-User Prios
condor_history	Completed Job Info
condor_submit_dag	Submit new DAG
condor_checkpoint	Force a checkpoint
condor_compile	Link Condor library

DMTCP & Parrot

- DMTCP (Checkpointing)
  - "Distributed MultiThreaded Checkpointing"
  - Developed at Northeastern University
  - **http://dmtcp.sourceforge.net/**
  - See Gene Cooperman's (Northeastern University) talk tomorrow (Wednesday) @ 4:05
- Parrot (Remote I/O)
  - Parrot is a tool for attaching existing programs to remote I/O system
  - Developed by Doug Thain (now at Notre Dame)
  - **http://www.cse.nd.edu/~ccl/software/parrot/**
  - dthain@nd.edu