Introduction to Condor

Steven Hosking

Massey University

Overview

- What is Condor
- What is used for
- How to use
- Hopfully a demo

About Condor

- University of Wisconsin, Madison
- Started in late 1980s
- Aim = develop a 'High Throughput Computing' software suite
- Can manage single PCs to Large Clusters
- Run on multiple systems (architectures/memory/OS)

What is HTC?

- HTC (High Throughput Computing) is running huge numbers of tasks over long periods of time
- Not short bursts of very fast computation (this is High Performance Computing)
- Difference? HPC is concernecd with operations/sec, HTC is concerned with operations/month or year even

What is Condor used for?

- Like many other grid/cloud projects, Condor is used to take advantage of otherwise unused computing power
- Single Machine – monitor jobs, pause if user uses machine, restart later
- Clusters/Pools – function as a cluster submission tool, if you cant afford a dedicated cluster, then everyday PCs can form a cluster and spare cycles can be 'scavenged' to speed up comutation
- Condor Flocking – connecting one Condor Pool to others
- Also Condor-G

Outline

- Can run on dedicated computing machines and 'sniff' for spare cycles on workstations
- i.e. we could run it at Massey, our Pool could entail Clusters (Helix2, BestGrid machine) and Lab workstations, but how?
- Condor ClassAds specify when a machine is available in the Pool and other information about it.

ClassAds

- Multiple systems (architectures/memory/OS) can be in a Pool
- How can our linux job run on a windows machine?

- ClassAds is the solution
- Each machine has a ClassAd

e.g. You may advertise that your machine is only willing to run jobs at night and when there is no keyboard activity on your machine for a specific time

- Each job has a ClassAd

e.g. You may specify your job needs at least 64mb and a INTEL architecture

```
:~$ condor_status

Name       Arch     OpSys        Mem(Mb)
Will       INTEL    WIN          128
Bob        INTEL    LINUX        512
Jane       SUN      SOLARIS      64
Steve      INTEL    WIN          1024

:~$ condor_status -l Steve
```

Will give complete machine ClassAd

Using Condor

- Similar to the PBS system we have been using i.e. submit a job to the Condor que
- But we must first choose a Universe to run in
- What is a Universe? Different runtime environments.
- Why? Condor has the ability to use idle workstations, what happens if a work station is taken control by user during a job? Universes come in.

Universes

- Main Universes (execution environment) in Condor

  * Standard - allows checkpointing, must relink with condor_compile
  * Vanilla - if condor_compile fails then use vanilla, no checkpointing
  * Parallel - for parallel running of jobs and programs written in MPI
  * Grid - submitting jobs to the remote Grids
  * Java - jobs written for the Java Virtual Machine

- Other Universes – Local, VM

Checkpointing
The ability to cause a job to vacate a non-idle workstation and migrate to an idle workstation is central to Condor's job scheduling.

Submitting Jobs

```
#include <stdio.h>

int main(void)
{
    printf("hello, Condor\n");
    return 0;
}
```

```
gcc -c hello.c -o hello.o
condor_compile gcc hello.o -o hello
```

```
########################
# Submit description file for hello program
########################
Executable     = hello
Universe       = standard
Output         = hello.out
Log            = hello.log
Queue
```

Will submit job once

```
condor_submit hello.submit
```

Massey University

Complex Submit Jobs

```
####################
# Use requirements
####################

Universe       =  vanilla
Executable     =  mathematica.$$(OpSys).$$(Arch)
Requirements   = (OpSys == "XP" && Arch =="INTEL") ||
                 (OpSys == "LINUX" && Arch =="INTEL")
Error   = mathematica.err
Input   = mathematica.in
Output  = mathematica.out
Log     = mathematica.log

Queue
```

Requires Compliled Files:
mathematica.XP.INTEL
mathematica.LINUX.INTEL

```
####################
# Use both requirements and rank
####################

Executable     = foo
Requirements   = Memory >= 32
Rank           = Memory >= 64

Error   = err.$(Process)
Input   = in.$(Process)
Output  = out.$(Process)
Log     = foo.log

Queue 150
```

Minimum

Prefered

150 input files

Submit 150 jobs

Parallel Submit Jobs

- Start a number of jobs at the same time
- Eg foo.submit

```
##########################################
##   submit description file for a parallel program
##########################################
universe = parallel
executable = foo
machine_count = 8
output = n_body.out
queue
```

Request 8 machines

- Condor will wait until all 'machine_count' machines of the same architecture and OS as host are available before starting the job.

Condor File Transfer Mechanism

- Condor has its own file sharing system, files can be transferred that are needed for a job.

- To enable file transfer:

```
should_transfer_files = YES | IF_NEEDED | NO
when_to_transfer_output = ON_EXIT | ON_EXIT_OR_EVICT
```

- What is transfered? Executable/Input and Output

- Note : default behavior of the file transfer mechanism varies across the different Condor universes, and it differs between UNIX and Windows

MPI Submit Jobs

- Supports MPICH
- n_body.submit

```
######################################
## Example submit description file
## for MPICH 1 MPI
## works with MPICH 1.2.4, 1.2.5 and 1.2.6
######################################
universe = parallel
executable = mp1script
arguments = n_body
machine_count = 4
should_transfer_files = yes
when_to_transfer_output = on_exit
transfer_input_files = n_body
queue
```

Request 4 machines

Codor file transfer commands

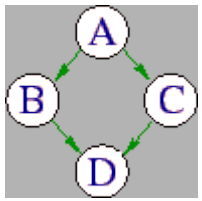
- Must use a extra script ('mp1script') to further define that framework
- A template script is supplied with Condor
- A bit more effort to setup but can be done

Other Condor Technologies

- Flocking
- DAGMan
- Condor-G
- Condor Pool Tools
- Condor File Transfer

DAGMan

- Also has a functionality called DAGMan
- Stands for... Directed Acyclic Graph Manager
- Which means... Condor can manages dependencies between jobs




```
# Filename: diamond.dag
#
JOB  A  A.submit
JOB  B  B.submit
JOB  C  C.submit
JOB  D  D.submit
PARENT A CHILD B C
PARENT B C CHILD D
```

```
# Filename: A.submit
#
executable = A
output = a.out
log = a.log
error a.error
```

```
condor_submit_dag diamond.dag
```

Flocking

- Allows one or more pools to be 'connected'
- If the job cannot be immediately run in the current pool then it can be 'flocked' to another pool to allow

Condor-G

- Also has the abilty to connect and share with Globus Project

Pool Tools

- Scripts that help manage a Condor Pool. The pool tools help you create a list of machines that should be in the pool, and to create a list that are in the pool.

Thankyou