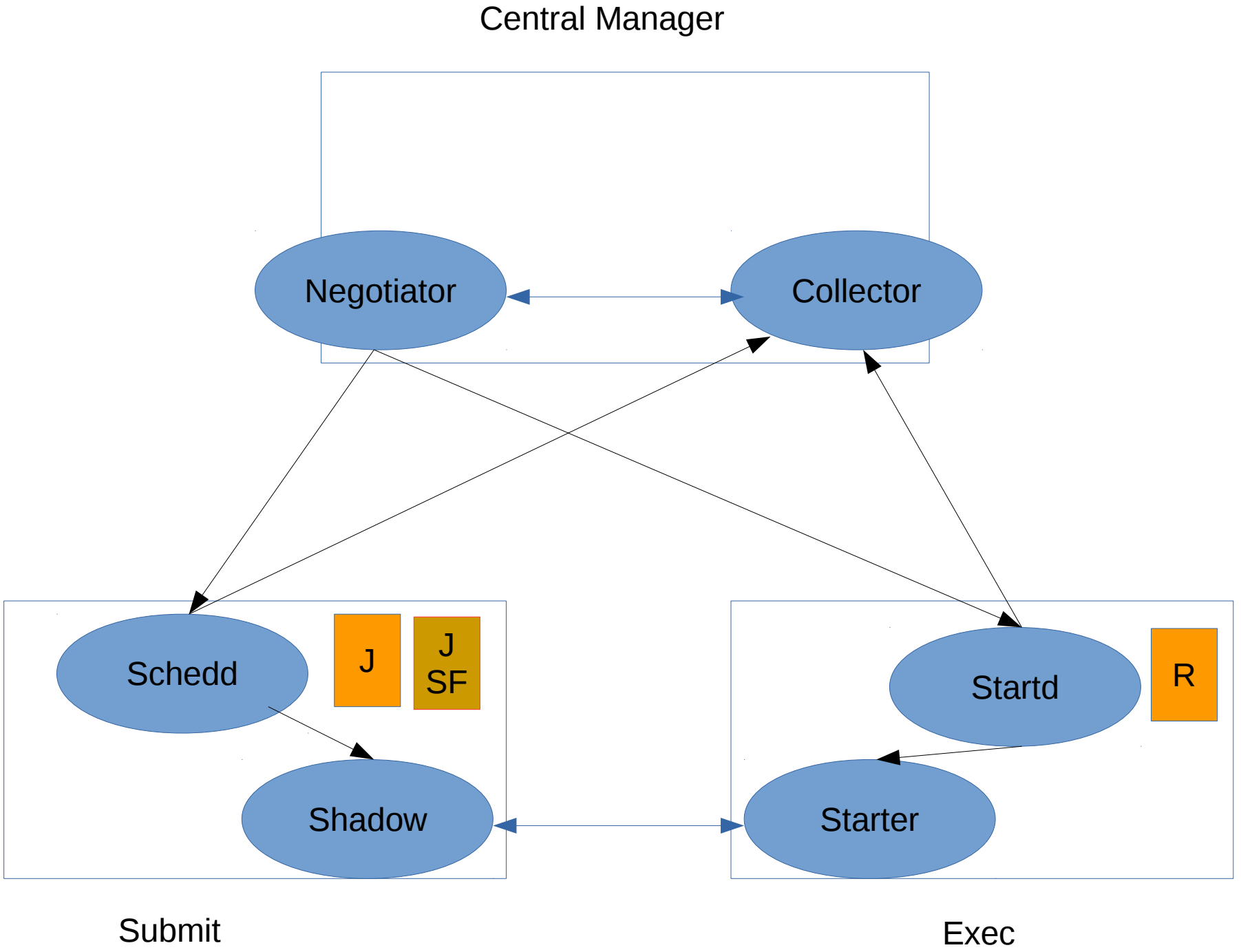
# HTCondor
# Admin Tutorial

- Come viene mandato un job in esecuzione remota
- Matchmaking,Chekponting
- ClassAds
- Policy
- Installazione
- Configurazione


Central Manager
Negotiator
Collector
Schedd
J
J
SF
Shadow
Startd
R
Starter
Submit
Exec

# ClassAd

- Il linguaggio usato per descrivere le informazioni su:
  - Job da eseguire (job ClassAd)
  - Macchine su cui eseguire i job (machine ClassAd)
- *Job ClassAd* e *machine ClassAd* devono essere compatibili (matching)

I job descrivono le richieste (obbligatorie):

```
Requirements = OpSys == "LINUX" &&
Arch == "INTEL"
```

e le preferenze:

```
Rank = Kflops
Rank = Memory
Rank = Memory >= 1000
```

[TRUE = 1.0, FALSE = 0.0]

Le *machine ClassAd* descrivono le caratteristiche della macchina:

```
OpSys                   = “LINUX”
Arch                    = “INTEL”
Disk                    = 500000
Memory                  = 32000
```

Le condizioni per eseguire job:

```
Requirements = TARGET.Owner==”rossi” || \
               (LoadAvg<0.3 && KeyboardIdle>15*60)
```

Le preferenze:

```
Rank = Friends + Group*10
```

# Machine ClassAd

```
Friends = (Owner == “rossi”) || (Owner
                == “verdi”)
Group = (Exp == “atlas”) || (Exp ==
             “cms”)
START = Friend && (Group || LoadAvg <
          0.3)
RANK = Friend + Group*10
```

# Matchmaking

Il matchmaker confronta *job ClassAds* e *machine ClassAd,* prendendo in considerazione `Requirements` (devono coincidere) e `Rank` (per definire la priorita') di entrambi

# Come esegue un job

- Si sceglie un universe (environment) per il job
- Il job deve essere batch-ready (dati di input accessibili)
- Si crea un submit description file
- (Si compila con condor_compile)
- I manda in coda con condor_submit

# I job Universe

- L'Universe descrive come vengono gestiti i job
- I principali sono:
  - **Standard**
  - **Vanilla**
  - Grid
  - Parallel
  - Java
  - Vm
  - Docker

# Standard Universe

- Permette di fare il checkpoint del job
- Richiede la ricompilazione del job con condor_compile
  - `$ condor_compile gcc myjob.c –o myjob`
- Remote system calls (I/O remoto)
  - Il job legge/scrive un file presente sulla *submit machine* come se fosse locale

# Chekpoint

- Quando una exec machine non e' in grado di continuare a eseguire un job, per es:
  - Il proprietario della macchina comincia a usarla
  - Il tempo dedicato al job finisce
  - Un job con priorita' maggiore chiede accesso alla macchina
- Viene creata una immagine del job con tutte le info necessarie per fare ripartire il job da dove si e' interrotto
  - Punto di esecuzione
  - Puntatori ai file di I/O, etc.

# Checkpoint

- L'immagine viene ricopiata sulla submit machine (o su un checkpoint server)
- Ricomincia la procedura di matchmaking
- Il job riparte su un'altra macchina
- Il job passa da macchina a macchina finche' non arriva al termine
- **High Throughput!**

# Vanilla Universe

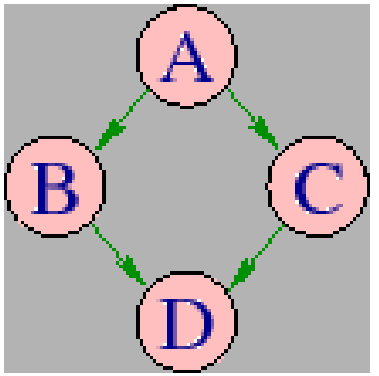
- Non richiede la ricompilazione del job
- Non permette il checkpoint
- Il tasferimento dei file di I/O deve essere specificato o essere su un file system condiviso tra submit ed exec machine (non si usano le remote system calls)

# DAGMan: esecuzione condizionata

- Si usa quando l'esecuzione di un (o piu') job deve avvenire dopo l'esecuzione di altri

DAG: Directed Acyclic Graph

```
JOB  A  A.condor
JOB  B  B.condor
JOB  C  C.condor
JOB  D  D.condor
PARENT A CHILD B C
PARENT B C CHILD D
```



# Esempio di Job Submit File

```
Universe   = standard
Executable = test
Input      = test.in
Output     = test.out
Log        = test.log
Error      = test.err
Queue
```

# Per eseguire piu' istanze di un job

```
Universe   = standard
Executable = test
Input      = test.in.$(process)
Output     = test.out.$(process)
Log        = test.log.$(process)
Error      = test.err.$(process)
Queue 100
```

# Il linguaggio dei ClassAd

- I parametri possono essere:
  - Integer, floating, logical, string
- I valori dei parametri si accedono con $

  B = $(A)
- L'ultimo valore e' quello che conta:

  A = 1

  B = $(A)

  A = 2 [B e' uguale a 2]

# Differenza tra =?= e ==, =!= e !=

L'operatore =?= produce risultati TRUE o FALSE, mentre == produce risultati TRUE, FALSE, UNDEFINED, o ERROR.

(10 == 10) ---------> TRUE

(10 == 5) ---------> FALSE

(10 == "ABC") ---------> ERROR

(10 == UNDEFINED) ---------> UNDEFINED

(UNDEFINED == UNDEFINED) -------→ UNDEFINED

# Differenza tra =?= e ==, =!= e !=

(10 =?= 10) —----------> TRUE

(10 =?= 5) —----------> FALSE

(10 =?= "ABC") —----------> FALSE

(10 =?= UNDEFINED) —----------> FALSE

(UNDEFINED =?= UNDEFINED) —-------> TRUE

# Espressioni

- Mettere parentesi!

  DIECI = 5+5

  CENTO = $(DIECI)*$(DIECI)

  CENTO diventa 5+5*5+5, cioe' 35

  DIECI = (5+5)

  CENTO = ($(DIECI)*$(DIECI))

  MILLE = ($(CENTO)*$(DIECI))

# Policy

- Chi deve eseguire un job, dove, quando e per quanto tempo?
- START
- RANK
- SUSPEND
- CONTINUE
- PREEMPT
- KILL

# START

- START e' la policy primaria
- Contiene parametri relativi alla macchina
  - carico della cpu
  - attivita' interattiva sulla tastiera
  - orario permesso per l'esecuzione di job
  - etc
- E parametri relativi al job
  - owner del job
  - macchina da cui e' stato lanciato
  - etc.

- START deve essere valutata come True perche' la exec machine esegua il job

# RANK

- Specifica le preferenze
- Numero floating point: piu' alto il numero maggiore il rank
  - RANK = kflops
  - RANK = (Memory > 8000)*10

# SUSPEND e CONTINUE

- Quando SUSPEND diventa `True` il job smette di essere eseguito ma rimane sulla *exec machine* (che risulta occupata)
- Quando CONTINUE diventa `True` il job riprende l'esecuzione da dove si era fermato

# PREEMPT e KILL

- Quando PREEMPT diventa `True` il job lascia la exec machine
  - Se si ha Universe = Standard il job fa un checkpoint
  - Se si ha Universe = Vanilla il job e' killed
- Quando KILL diventa `True` il job e' killed (se un checkpoint e' gia' iniziato viene interrotto)

# Minimal setting

- Per fare in modo che una macchina possa eseguire sempre un job

```
START = True
RANK = 0
SUSPEND = False
CONTINUE = True
PREEMPT = False
KILL = False
```

# Policy per “gruppi”

- Un esperimento (ATLAS) aggiunge macchine in un pool, ma ne vuole avere la priorita'
  - Se un utente ATLAS e uno CMS hanno entrambi un job in coda, viene eseguito quello di ATLAS
  - Se un job di CMS e' gia' in esecuzione e entra in coda uno di ATLAS, quello di CMS lascia la macchina (checkpoint)

# Policy per macchine di ATLAS

```
START = True
RANK = Experiment =?= “ATLAS”
SUSPEND = False
PREEMPT = False
KILL = False
```

# Job ClassAds per ATLAS

- Si possono aggiungere parametri custom mettendo “+” come prefisso

```
Universe = Standard
Executable = simul1
+Experiment = “ATLAS”
Queue
```

# Policy piu' complesse

- Si vuole rendere massima priorita' ad alcuni utenti (antonio,luigi), seguita dall'esperimnto ATLAS, seguita dall'esperimento CMS
  - Si usa il paramtero `Owner` che e' definito automaticamente

```
IsOwner = (Owner == “antonio”) \
        || (Owner == “luigi”)
IsAtlas = (Experiment =?= “ATLAS”)
IsCms = (Experiment =?= “CMS”)
RANK = $(IsOwner)*20 + $(IsAtlas)*10 + \
       $(IsCms)
```

# Ancora Policy

- Si voglio runnare job solo per 12 ore
- Eccezione per ATLAS: i job girano per 24 ore

# Parametri interni utili

- CurrentTime
  - Definisce il tempo attuale in “Unix epoch time” (secondi dall' 1 gennaio 1970, ore 00:00)
- EnteredCurrentActivity
  - Definisce il tempo in cui HTCondor ha iniziato l'attivita' (sempre in Unix epoch time)

```
ActivityTimer = (CurrentTime - \
          EnteredCurrentActivity)

ORA = (60*60)

DODICIORE = ($(ORA)*12)

GIORNO = ($(ORA)*24)

PREEMPT = ($(IsAtlas) && \

    ($(ActivityTimer) > $GIORNO)) || \

  (!$(IsAtlas) && ($(ActivityTimer) >
       $DODICIORE))
```

# Condor sui pc personali

- Comportamento *opportunistico*
- Condor puo' utilizzare risorse quando queste sono libere

# Definizione di “risorse libere”

- Non c'e' attivita' del mouse o tastiera per 5 minuti
- Il load average (carico medio della CPU) e' < 0.3

# Cosa il PC dovrebbe fare

- START  quando il pc diventa libero
- SUSPEND  appena una attivita' e' rilevata
- PREEMPT (chekpoint)  se l'attivita' continua per 5 minuti
- KILL  se impiega piu' di 5 minuti per fare il PREEMPT

# Parametri utili

- LoadAvg
  - Il carico della CPU totale
- CondorLoadAvg
  - Il carico della CPU dovuto a HTCondor
- KeyboardIdle
  - Secondi trascorsi dall'ultima attivita' di tastiera o mouse

# Macro da usare

- NonCondorLoadAvg = (LoadAvg – CondorLoadAvg)
- LoadMax = 0.3
- CPU_Busy = ($(NonCondorLoadAvg) >= $(LoadMax))
- CPU_Idle = (!$(CPU_Busy))
- Keyboard_Busy = (KeyboardIdle < 10)
- Keyboard_Idle = (KeyboardIdle > 300)
- Machine_Busy = ($(CPU_Busy) || $(Keyboard_Busy))

# Desktop Machine Policy

- START = $(CPU_Idle) && \
  $(Keyboard_Idle)
- SUSPEND = $(Machine_Busy)
- CONTINUE = $(CPU_Idle) && \
  KeyboardIdle > 120
- PREEMPT =(Activity == "Suspended" \
  && $(ActivityTimer) > 300
- KILL = $(ActivityTimer) > 300

# Controllo dello stato del pool

```
# condor_status

Name                OpSys      Arch      State     Activity LoadAv Mem   ActvtyTime

slot1@ckptmi.mi.in LINUX      INTEL  Unclaimed Idle      0.000 1013  0+04:55:10
slot2@ckptmi.mi.in LINUX      INTEL  Unclaimed Idle      0.000 1013 31+07:16:34
slot1@condor.pv.in LINUX      INTEL  Unclaimed Idle      0.000  473  6+11:16:02
slot2@condor.pv.in LINUX      INTEL  Unclaimed Idle      0.000  473  6+11:16:24
slot1@icascan4.pv. LINUX      INTEL  Owner     Idle      0.400  790  0+01:55:04
slot2@icascan4.pv. LINUX      INTEL  Owner     Idle      0.000  790  0+01:55:05
```

# Parametri di `condor_status`

- `-l nomemacchina`
  - mostra tutti i ClassAd di una macchina
- `-constraint`
  - Mostra I ClassAd che soddisfano un vincolo

```
condor_status -constraint 'Memory >= 1024 && \
                Activity == "Idle"'
```

# Controllo dello stato delle code

$ condor_q

-- Schedd: condor03.bo.infn.it : <131.154.12.171:32554?...

| ID | OWNER | SUBMITTED | RUN_TIME | ST | PRI | SIZE | CMD |
|---|---|---|---|---|---|---|---|
| 12.0 | semeria | 11/24 15:51 | 0+00:00:01 | R | 0 | 2.4 | gethost |
| 12.1 | semeria | 11/24 15:51 | 0+00:00:01 | R | 0 | 2.4 | gethost |
| 12.2 | semeria | 11/24 15:51 | 0+00:00:00 | R | 0 | 2.4 | gethost |
| 12.3 | semeria | 11/24 15:51 | 0+00:00:00 | R | 0 | 2.4 | gethost |
| 12.4 | semeria | 11/24 15:51 | 0+00:00:00 | I | 0 | 2.4 | gethost |

# Parametri di `condor_q`

- `-long job_id`
  - Mostra tutti I ClassAd di un job
- `-analyze job_id`
  - Mostra su quali macchine puo' girare il job e su quali no

# condor_q -analyze

[semeria@condor04 ~]$ condor_q -analyze 7.0

007.000:  Run analysis summary.  Of 38 machines,
30 are rejected by your job's requirements
0 reject your job because of their own requirements
0 match and are already running your jobs
0 match but are serving other users
8 are available to run your job

The Requirements expression for your job is:

( TARGET.Machine is "condor02.bo.infn.it" ) && ...

# Comandi principali

- `condor_status` View Pool Status
- `condor_q` View Job Queue
- `condor_compile` Link HTCondor library with job
- `condor_submit` Submit new Jobs
- `condor_rm` Remove Jobs

# Installazione Linux

- RedHat repository

cd /etc/yum.repos.d

wget
http://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-rhel6.repo

yum -y install condor

- Debian repository

echo "deb http://research.cs.wisc.edu/htcondor/debian/stable/ squeeze contrib"
>> /etc/apt/sources.list

apt-get update

apt-get install condor

# Installazione Mac

- Download

http://research.cs.wisc.edu/htcondor/downloads/

tar -zxvf condor-8.4.2-x86_64_MacOSX-stripped.tar.gz

cd condor-8.4.2-x86_64_MacOSX7-stripped

./condor_configure –install –make-personal-condor

[modificare il file etc/condor_config. “etc/” all'interno della directory]

./sbin/condor_init

./sbin/condor_master

./sbin/condor_startd

./sbin/condor_schedd

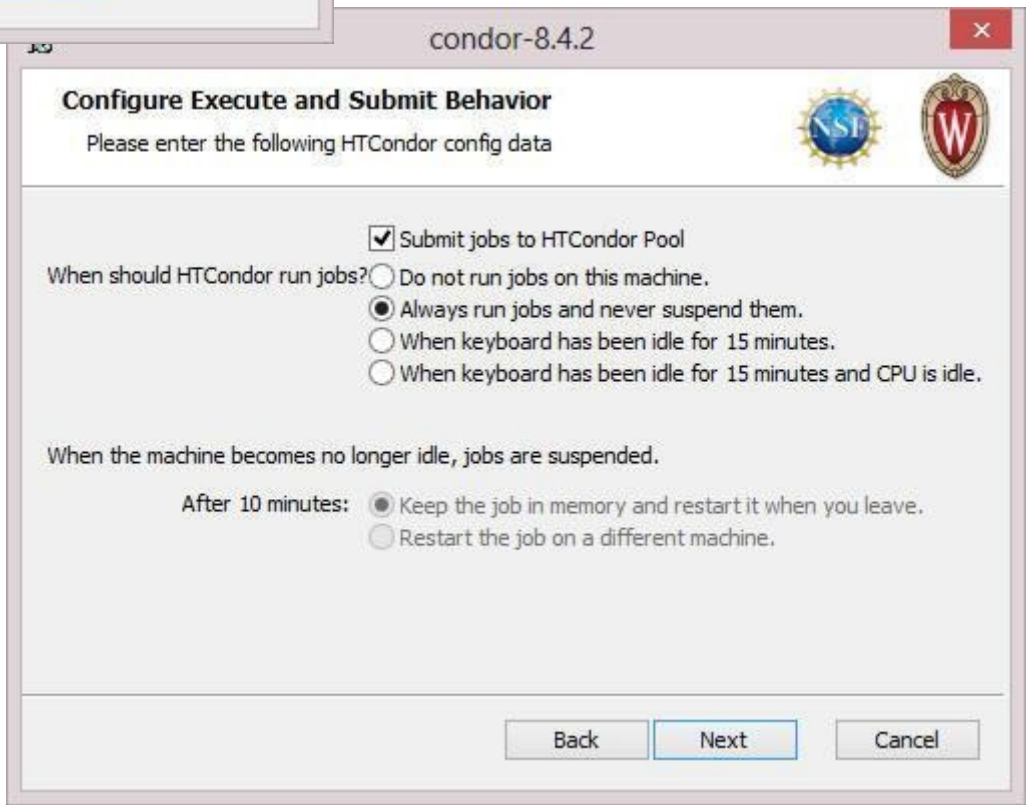
source condor.sh

# Installazione Windows

- Download http://research.cs.wisc.edu/htcondor/downloads/

  condor-8.4.2-349384-Windows-x86.msi
- Installazione grafica (abbastanza) autoesplicativa


condor-8.4.2 Setup
Welcome to the condor-8.4.2 Setup Wizard
The Setup Wizard will install condor-8.4.2 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.
HTCondor
Back
Next
Cancel


condor-8.4.2
Configure Execute and Submit Behavior
Please enter the following HTCondor config data
Submit jobs to HTCondor Pool
When should HTCondor run jobs?
Do not run jobs on this machine.
Always run jobs and never suspend them.
When keyboard has been idle for 15 minutes.
When keyboard has been idle for 15 minutes and CPU is idle.
When the machine becomes no longer idle, jobs are suspended.
After 10 minutes:
Keep the job in memory and restart it when you leave.
Restart the job on a different machine.
Back
Next
Cancel

# File di configurazione principale

- /etc/condor/condor_config

```
##  This is the global configuration file for condor. This is where
##  you define where the local config file is. Any settings
##  made here may potentially be overridden in the local configuration
##  file.  KEEP THAT IN MIND!  To double-check that a variable is
##  getting set from the configuration file that you expect, use
##  condor_config_val -v <variable name>
```

# condor_config file

```
##  Where have you installed the bin, sbin and lib condor directories?
RELEASE_DIR = /usr
##  Where is the local condor directory for each host?  This is where the local config file(s), logs and
##  spool/execute directories are located. this is the default for Linux and Unix systems.
LOCAL_DIR = /var
##  Pathnames
RUN     = $(LOCAL_DIR)/run/condor
LOG     = $(LOCAL_DIR)/log/condor
## Central Manager
CONDOR_HOST = condor01.bo.infn.it
## Daemons running (specify machine type)
#DAEMON_LIST = MASTER,COLLECTOR, NEGOTIATOR, SCHEDD, STARTD
DAEMON_LIST = MASTER, SCHEDD, STARTD
```

# Configurazione locale

- /etc/condor/condor_config.local

```
## Always run jobs
START           = True
SUSPEND         = False
CONTINUE        = True
PREEMPT         = False
KILL            = False
```

# Log files

- Controllare i log file in /var/log/condor
  - MasterLog
  - SchedLog
  - StarterLog
  - NegotiatorLog
  - CollectorLog

# Ringraziamenti


- Daniela Bortolotti
- Patrizia Calligola

# Operatori aritmetici

The operators *, /, + and - operate arithmetically only on integers and reals.

Arithmetic is carried out in the same type as both operands, and type promotions from integers to reals are performed if one operand is an integer and the other real.

If either operand is not a numerical type, the value of the operation is ERROR.

# Operatori logici

The logical operators && and || operate on integers and reals. The zero value of these types are considered FALSE and non-zero values TRUE.

Any string operand is equivalent to an ERROR operand for a logical operator. In other words, TRUE && "foobar" evaluates to ERROR

# Operatori di comparazione

The comparison operators ==, !=, <=, <, >= and > operate on integers, reals and strings.

Comparisons are carried out in the same type as both operands, and type promotions from integers to reals are performed if one operand is a real, and the other an integer. Strings may not be converted to any other type, so comparing a string and an integer or a string and a real results in ERROR.

String comparisons are case insensitive for most operators. The only exceptions are the operators =?= and =!=, which do case sensitive comparisons assuming both sides are strings.