Un'introduzione a

Condor
High Throughput Computing

Condor Project
Computer Sciences Department
University of Wisconsin-Madison
condor-admin@cs.wisc.edu
http://www.cs.wisc.edu/condor

Incontriamo Frieda

Frieda è uno scienzato, ed ha un **grande** problema.

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

Il problema di Frieda

Valutazione di una funzione assegnata:

F(x,y,z) per

20 valori di x

10 valori di y

3 valori di z

20 x 10 x 3 = 600 combinazioni!

- La valutazione di **F** su una tipica workstation impiega circa 6 ore

  600 runs X 6 ore = 3600 ore

- **F** richiede un moderato carico di memoria

  256 Mbytes

- I dati di Input ed Output (I/O) sono rispettivamente:

  (x,y,z) - 5 MBytes

  F(x,y,z) - 50 MBytes

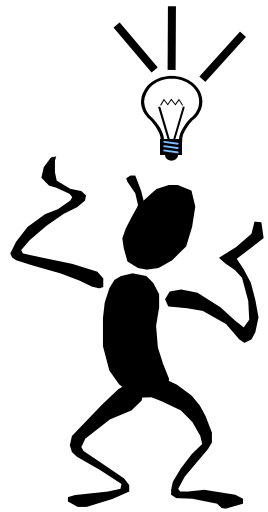
Applicazioni parameter sweep

- L'applicazione di Frieda è di tipo parameter sweep
- Il programma esegue lo stesso codice con diversi insiemi di parametri di input
- Queste applicazioni sono naturalmente parallele

Frieda ha 600 simulazioni da eseguire su un calcolatore...

Come aiutarla?

SOLUZIONE!!



Frieda ha bisogno di un

batch processing system

Ovvero di un esecutore di sequenze di programmi senza interazione con l'utente

Esempi di batch processing systems

- PBS (Portable Batch System) and Open PBS
- LSF (Load Sharing Facility)
- Sun Grid Engine
- Slurm
- Condor

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

Caratteristiche di Condor

- High throughput computing
- Può essere configurato in diversi modi
- Supporta meccanismi di sicurezza
- Possiede una buona interoperabilità con molti tipi di Griglie computazionali e gli altri batch processing systems

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

High-Troughput Computing

- High-Throughput Computing (HTC)
  l'uso di diverse risorse computazionali per un lungo periodo di tempo (mesi, anni)
- High Performance Computing (HPC)
  l'uso di una grossa potenza di calcolo per un breve periodo di tempo (giorni, ore)

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

High-Troughput Computing

- Gli utenti HTC sono interessati a quante operazioni sono possibili per mese o per anno
- Le prestazioni di ambienti HPC sono spesso misurate in termini di Floating Point Operations Per Seconds (FLOPS)

High-Troughput Computing

- L'utente HTC è interessato alla possibilità di conoscere quante esecuzioni possono essere effettuate in un periodo di tempo lungo, piuttosto che quanto veloce debba essere la singola esecuzione

JOB

- Con il termine Job indichiamo la singola esecuzione di un programma
- Condor è un sistema di gestione di JOB

Una semplice istallazione di Condor permetterà ...

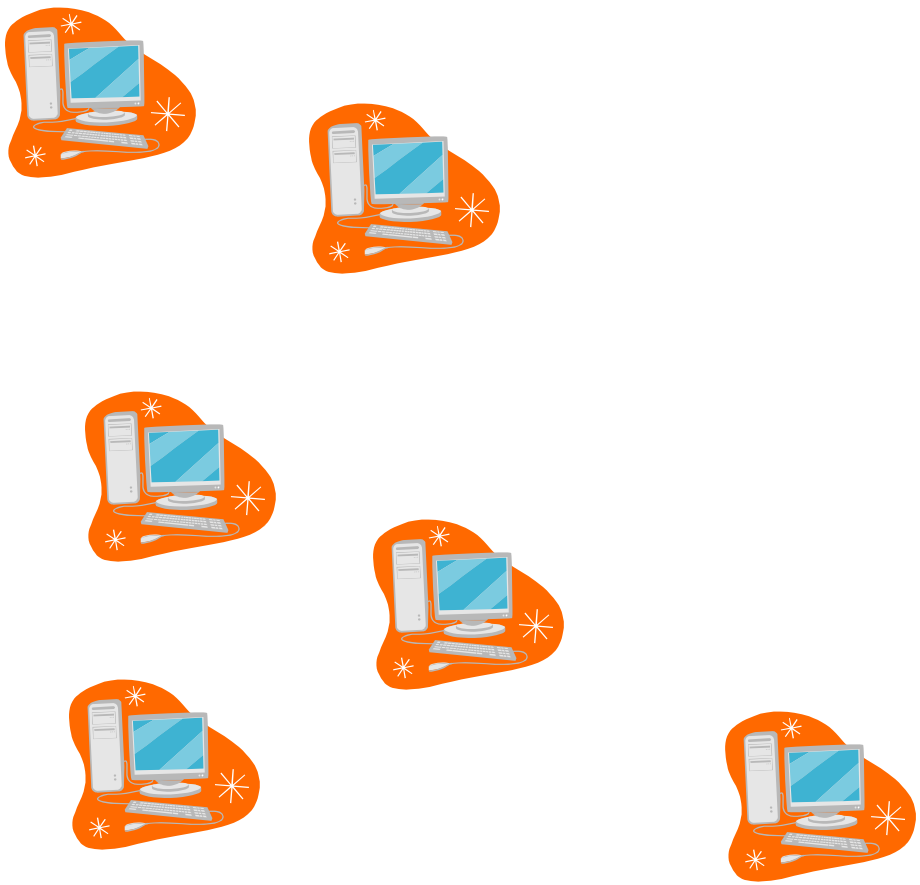
- L'esecuzione di un ordine prestabilito di insiemi di job
- L'osservazione dei job e la notifica sul loro stato di avanzamento
- La descrizione delle attività dei job in opportuni "file di Log"
- Un meccanismo di fault tolerance per i job
- La definizione di una politica (policy) sulla propria workstation per l'esecuzione di job (come e quando utilizzare la propra workstation per l'esecuzione)

Pertanto, con Condor…

Le 600 simulazioni di Frieda diventeranno:

600 Condor job,
descritti da 1 file di testo,
sottomessi con 1 comando

un Condor pool

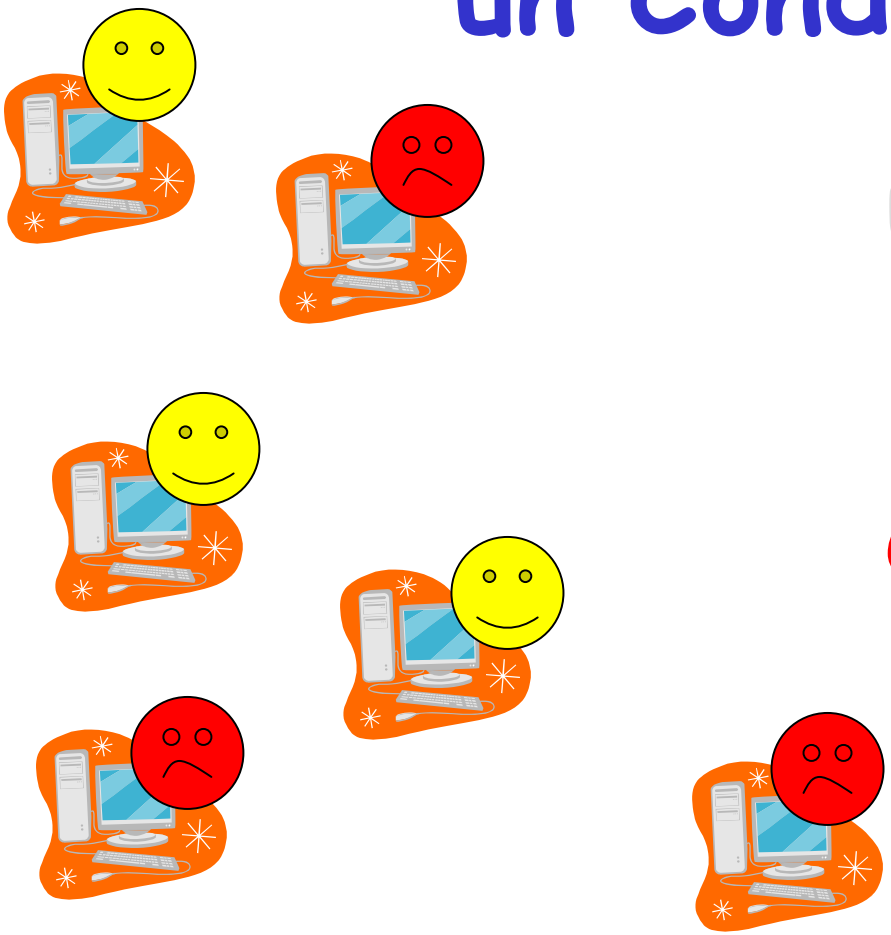


Uno o più calcolatori su cui è in esecuzione Condor

Ogni calcolatore definisce la propria politica per l'esecuzione di jobs

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

un Condor pool



Una o più calcolatori su cui è in esecuzione Condor

Ogni calcolatore definisce la propria politica per l'esecuzione di jobs

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

Come funziona il Matchmaking di Condor

- I job e i calcolatori hanno ciascuno le proprie richieste e le proprie preferenze

- Condor mette insieme (effettua match) job e calcolatori sulla base delle richieste e delle preferenze

Job

Preferenze e requisiti di un Job:

- Una piattaforma Linux/x86
- Il calcolatore con maggior memoria
- Una macchina che si trova preferibilmente nel dipartimento di chimica

Calcolatore

Preferenze e requisiti di un Calcolatore:

- Esecuzione di jobs solo quando non c'è attività sulla tastiera
- I job di Frieda hanno la precedenza sugli altri
- Il calcolatore appartiene al dipartimento di Fisica
- Non vengono eseguiti job appartenenti al Dr. Rossi

Prima di cominciare:
Come sottomettere Job a Condor

1. Scegliere un opportuno universo per il job
2. Preparare il job all'esecuzione nel batch-system
3. Definire il submit description file
4. Eseguire il comando condor_submit per inserire i job in coda

1. Scegliere un Universo

- L'Universo definisce un determinato modo di gestire i job (individuando già alcune sue caratteristiche)
- Alcuni universi:
  - vanilla
  - standard
  - grid
  - java
  - mpi
  - scheduler
  - local

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

L'universo Vanilla

- Per job seriali
- Il nome viene dal fatto che, come il gelato alla vaniglia, può essere utilizzato per (quasi) tutti i job
- Non fornisce servizi avanzati ma non pone restrizioni

L'universo standard

- Fornisce alcuni serivizi
  - accesso all'I/O remoto
  - fault-tolerance
  - checkpoint
- Presenta restrizioni sui programmi che possono essere eseguiti

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

L'universo PVM

- Per Programmi paralleli che utilizzano l'interfaccia Parallel Virtual Machine

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

L'universo Parallel

- Supporta vari ambienti di programmazione parallela tra cui MPI
- Supporta job che devono essere co-schedulati
- Rimpiazza l'universo MPI

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

L'universo Grid

- Consente all'utente di utilizzare tecnologie Grid o sistemi di batch diversi da Condor per l'esecuzione di job
- Supporta vari tipi di teconologie:
  - Globus (GT2, GT3, GT4), Unicore, Nordugrid, PBS, LSF.

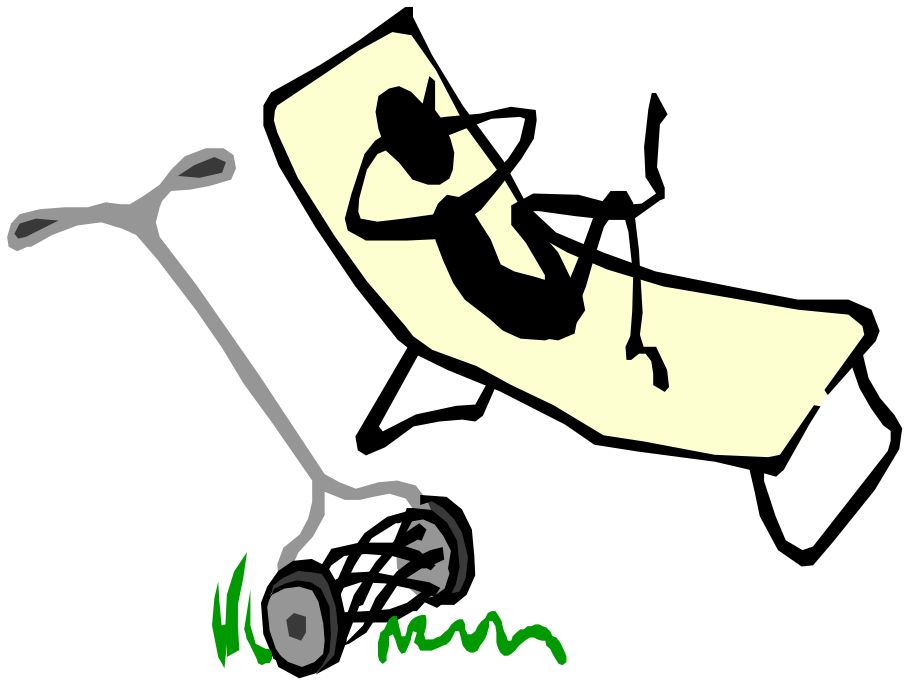
http://www.cs.wisc.edu/condor

L'universo Local

- Indica a Condor di non effettuare match tra Job ed i calcolatori disponibili
- Il Job viene eseguito sul calcolatore su cui è stato sottomesso
- Il Job non viene mai sospeso

2. Preparare il job all'esecuzione nel batch-system

- L'esecuzione del job avviene in background, pertanto, non deve necessitare di:
  - Input interattivi
  - Finestre
  - GUI



2. Preparare il job all'esecuzione nel batch-system

- Può utilizzare STDIN, STDOUT, e STDERR (la tastiera ed il video), ma questi sono files (non dispositivi).

Come ad esempio, in ambiente Unix

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

3. Definire il submit description file

- Un file di testo ASCII
- Il suo nome e la sua estensione sono irrilevanti
- Descrive il job per l'esecuzione su Condor
- Può descrivere più jobs alla volta, ognuno con differenti input, differenti argomenti, differenti output, etc.

Descrizione del Job

Il submit description file può contenere:

- I nomi dei file di input e di output
- Gli argomenti di linea di comando
- Variabili di ambiente
- Requisiti
- Preferenze (dette rank)

Esempio di Submit Description File

```
# il nomde del file è sim.submit
# (Le linee di commento cominciano con #)

universe   = vanilla
executable = sim.exe
output     = output.txt
queue

# NOTA: le parole che figurano sulla sinistra  non sono
#       case sensitive, mentre i nomi dei file si!
```

4. Eseguire il comando condor_submit

- Il comando condor_submit richiede il nome del submit description file:

```
condor_submit sim.submit
```

- condor_submit quindi
  - verifica se ci sono errori nel submit description file
  - definisce una ClassAd che descrive il job (o i job)
  - posiziona il job (o i job) in coda

ClassAds

- E' una rappresentazione di dati interna a Condor

  E' simile ad un annuncio economico

CERCASI:

Cane, colore scuro, taglia piccola. Prezzo max € 15.

- Ogni ClassAd può avere molti attributi
- Rappresenta un oggetto ed i suoi attributi

ClassAds

Le ClassAds contengono

- Informazioni:
  - il job eseguibile è `analysis.exe`
  - il carico medio della macchina è 5.6
- Richieste di opportuni requisiti:
  - E' richiesta una macchina con Linux
- Descrizione di preferenze:
  - Questa macchina preferisce eseguire jobs provenienti dal gruppo di fisica

ClassAds

Le ClassAds sono:

- semi-strutturate
- estensibili dall'utente
- schema-free
- Il formato è:

Attributo = Espressione

Esempio:

```
MyType       = "Job"        ←— Stringa
TargetType   = "Machine"
ClusterId    = 1377         ←— Numero
Owner        = "roy"
Cmd          = "sim.exe"
Requirements =              ←— Boolean
   (Arch == "INTEL")
&& (OpSys == "LINUX")
&& (Disk >= DiskUsage)
&& ((Memory * 1024)>=ImageSize)
…
```

Cercasi cane…

ClassAd della "risorsa"

```
Type = "Cane"
Color = "Scuro"
Price = 12
```

ClassAd del "Job"

```
...
 Requirements =
  (type == "Cane") &&
  (color == "Scuro") &&
  (price <= 15)
...
```

La Coda dei Job

- Il comando condor_submit invia la ClassAd(s) del job alla coda
- I Job entrano in coda con un'operazione
  - atomica: consegna in due fasi
  - ben definita: senza fallimenti parziali
- Il comando condor_q permette di visualizzare la coda dei job

Esempio di condor_submit e condor_q

```
% condor_submit sim.submit
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 sim.exe

1 jobs; 1 idle, 0 running, 0 held

%
```

Analisi completa della ClassAd

```
% condor_q -l

-- Submitter: perdita.cs.wisc.edu :
   <128.105.165.34:1027> :
MyType = “Job”
TargetType = “Machine”
ClusterId = 1
QDate = 1150921369
CompletionDate = 0
Owner = “frieda”
RemoteWallClockTime = 0.000000
LocalUserCpu = 0.000000
LocalSysCpu = 0.000000
RemoteUserCpu = 0.000000
RemoteSysCpu = 0.000000
ExitStatus = 0
…
```

I File di Input, Output ed Error

```
universe   = vanilla
executable = sim.exe
input      = input.txt
output     = output.txt
error      = error.txt
log        = sim.log
queue
```

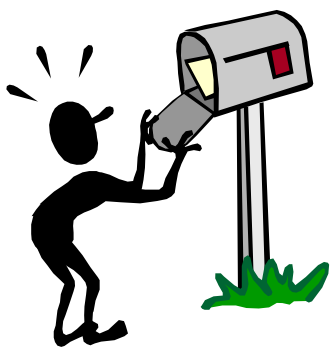
File di tipo testo contenente lo standard input

File di tipo testo contenente lo standard output

File di tipo testo dove viene reindirizzato lo standard error

Come controllare i job

- Condor può spedire un e-mail sull'evento all'utente che ha sottomesso il job
- Nel submit description file bisogna specificare quando si desidera ricevere tali e-mail



```
Notification = Never
             = Error
             = Always    <-- di default
             = Complete
```

Come controllare i job

- Definire un log per ogni evento del job
- Bisogna specificare il nome del file di log, nel submit description file:

```
log = sim.log
```

- Tale file descrive tutta La storia di un Job

Esempio di un file di Log

```
000 (0001.000.000) 05/25 19:10:03 Job submitted from host:
<128.105.146.14:1816>

...

001 (0001.000.000) 05/25 19:12:17 Job executing on host:
<128.105.146.14:1026>

...

005 (0001.000.000) 05/25 19:13:06 Job terminated.
    (1) Normal termination (return value 0)

...
```

Identificazione dei Job

- Ogni job in esecuzione in ambiente condor è detto **Processo**
- Un insieme di più processi mandati in esecuzione con un solo submission file e' detto **Cluster**

Come identificare cluster e processi?

Codice di identificazione dei Job

L'identificativo del processo (inizia sempre da 0)

3.0

cluster.process

2.6

cluster.process

Ogni numero cluster.process è detto identificativo del job (job ID)

Un altro es.di Submit Description File per Frieda

```
# Esempio di un cluster con 2 processi
Universe   = vanilla
Executable = analyze
Input      = a1.in
Output     = a1.out
Error      = a1.err
Log        = a1.log
Queue
Input      = a2.in
Output     = a2.out
Error      = a2.err
Log        = a2.log
Queue
```

Esempio di sottomissione dei primi Job di Frieda

```
% condor_submit a.submit
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   4/15 06:52  0+00:02:11 R  0   0.0  sim.exe
  2.0    frieda   4/15 06:56  0+00:00:00 I  0   0.0  analyze
  2.1    frieda   4/15 06:56  0+00:00:00 I  0   0.0  analyze
3 jobs; 2 idle, 1 running, 0 held
%
```

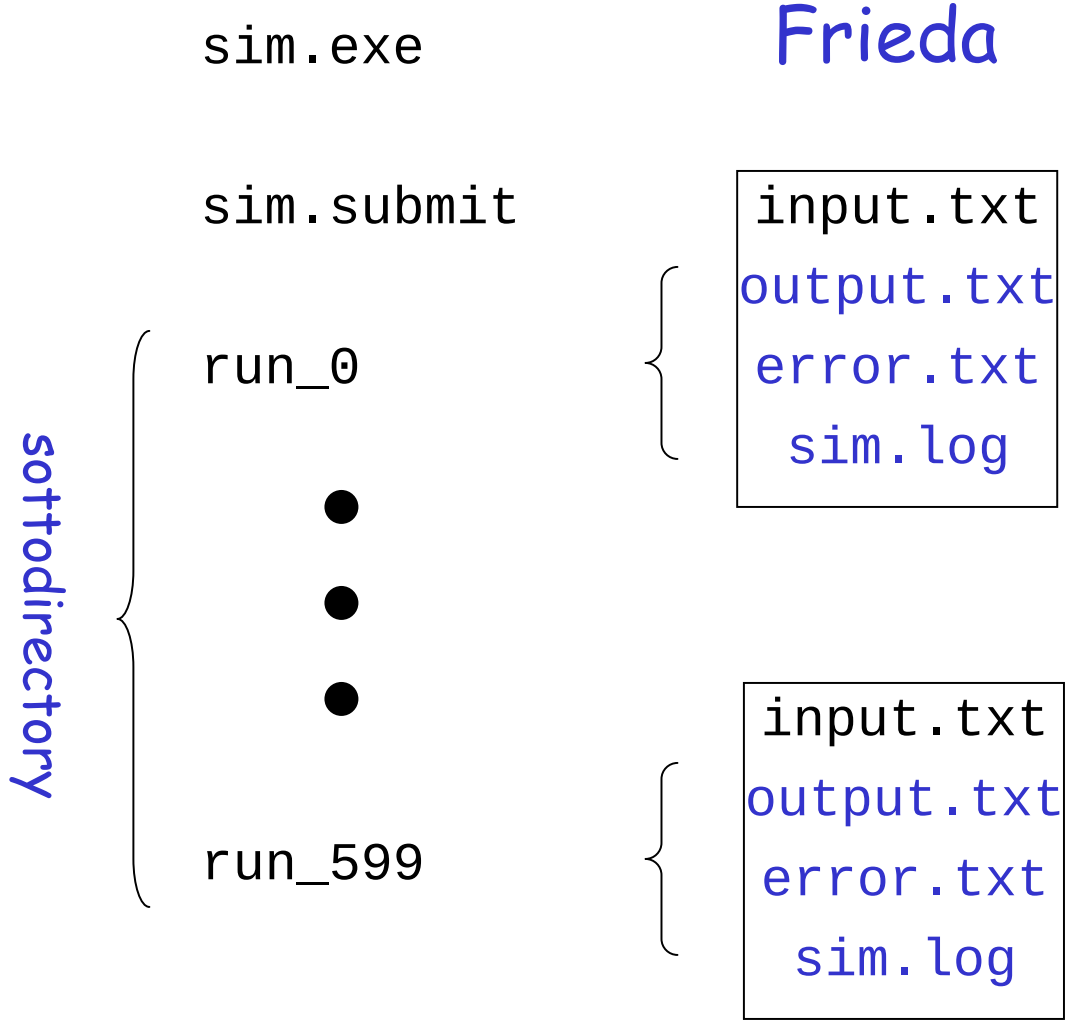
MA…
I job di Frieda sono 600!

- Mettere tutti i file di `input`, di `output`, di `error` e di `log` dei 600 job in una directory
  - Un file per ciascun tipo per ogni job
  - 4 files x 600 jobs = 2400 files !!

Oppure..

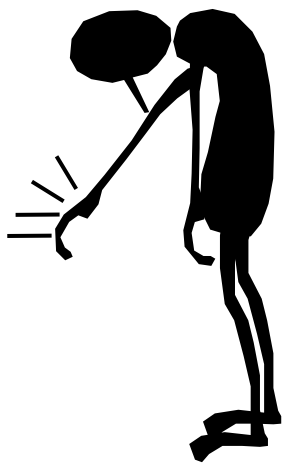
- Definire una sottodirectory per ciascun job
  - 600 directory

Esempio di sottodirectory per I job di Frieda



Description submit file dei 600 Job di Frieda

```
Universe   = vanilla
Executable = sim.exe
Input      = input.txt
Output     = output.txt
Error      = error.txt
Log        = sim.log
InitialDir = run_0
Queue
InitialDir = run_1
Queue
    .  .  .
InitialDir = run_599
Queue
```



600 ripetizioni!!

Il Submit Description file è

Troppo grande

circa 1200 linee

da compilare a mano…

Utilizzo della macro $(Process)

$(Process) assume

il valore dell'identificativo del processo

Description submit file dei 600 Job di Frieda

```
Universe   = vanilla
Executable = sim.exe
Input      = input.txt
Output     = output.txt
Error      = error.txt
Log        = sim.log
InitialDir = run_0
Queue
InitialDir = run_1
Queue
    . . .
InitialDir = run_599
Queue
```

```
Universe   = vanilla
Executable = sim.exe
Input      = input.txt
Output     = output.txt
Error      = error.txt
Log        = sim.log
InitialDir = run_$(Process)
Queue 600
```

Sottomissione dei 600 job di Frieda …

```
% condor_submit sim.submit
Submitting job(s)
   ..................................................
   ..................................................
   ..................................................
   ..................................................
   ..................................................
   ..............
Logging submit event(s)
   ..................................................
   ..................................................
   ..................................................
   ..................................................
   ..................................................
   ..............
600 job(s) submitted to cluster 3.
```

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

…e monitoraggio della coda

```
% 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
3.0    frieda  4/20 12:08 0+00:00:05 R  0   9.8  sim.exe
3.1    frieda  4/20 12:08 0+00:00:03 I  0   9.8  sim.exe
3.2    frieda  4/20 12:08 0+00:00:01 I  0   9.8  sim.exe
3.3    frieda  4/20 12:08 0+00:00:00 I  0   9.8  sim.exe
...
3.598  frieda  4/20 12:08 0+00:00:00 I  0   9.8  sim.exe
3.599  frieda  4/20 12:08 0+00:00:00 I  0   9.8  sim.exe

600 jobs; 599 idle, 1 running, 0 held
```

Gli argomenti a linea di Comando

- Gli argomenti che devono essere specificati a linea di comando, ad es:

```
% sim.exe 26 100
```

Devono essere inseriti nel submit description file:

```
Executable = sim.exe
Arguments = ”26 100”
```

Oppure si può utilizzare la macro $(…)

```
Arguments = ”$(Process) 100”
```

Rimuovere i job dalla coda

- Il comando condor_rm rimuove un job o un insieme di job dalla coda

- Ogni utente può rimuovere SOLO i propri job
  - L'utente root di Unix o l'administrator di Windows possono rimuovere qualsiasi job

Rimuovere i jobs dalla coda

- Per cancellare uno o più job si deve specificare il suo jobID:

```
condor_rm 4.1
```

(cancella il processo 1 del cluster 4)

```
condor_rm 4
```

(cancella tutti i processi del cluster 4)

- Per cancellare TUTTI i job:

```
condor_rm -a
```

Installazione

- Un primo passo: un personal Condor

Frieda comincia con un Personal Condor

- Condor è in escuzione sulla workstation di Frieda
- Non sono richiesti accessi da parte di root / administrator
- Non si necessita di un amministratore di sistema
- Dopo l'installazione, Frieda sottomette i suoi job al suo Personal Condor…

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

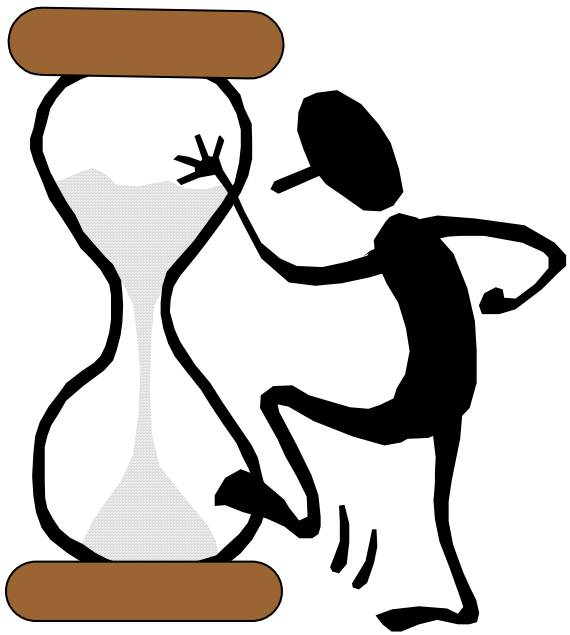
Download di Condor

- Condor è un software free, disponibile all'indirizzo:

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

- Sono disponibili versioni per la propria piattaforma (sistema operativo e architettura)
  - disponibile per diverse piattaforme Unix (inclusi Linux e Apple's OS/X)
  - disponibile per Windows NT / XP

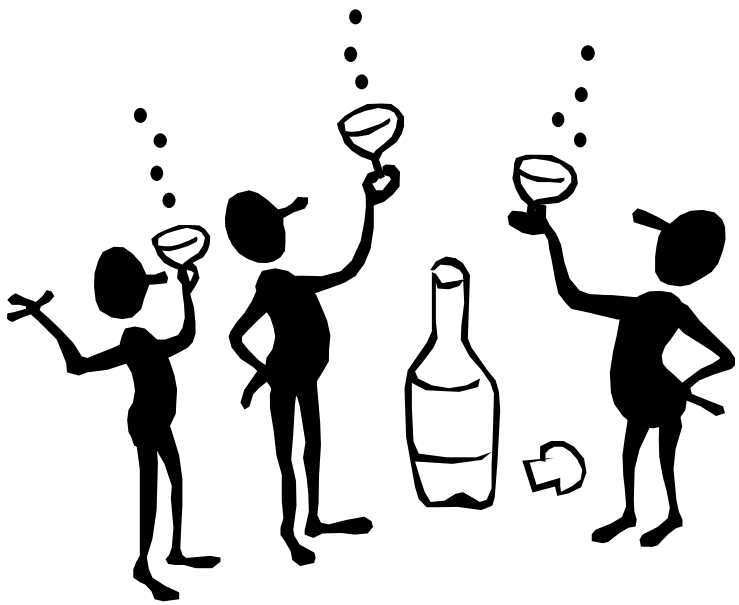
Il Personal Condor…
Bello! Ma… non sufficiente!



I 600 job di Frieda,

necessitano ciascuno di 6 ore di esecuzione per un totale di almeno 150 giorni ! (Se i job sono eseguiti 24 ore su 24..)

IDEA!



Frieda ha molti amici (con computer!): Fred, Ford, Fiona, and Francine.

Ognuno può installare Condor!!!

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

Gli amici di Frieda: un Condor Pool

Fred:

permette esecuzioni **solo** di notte

Fiona:

Non esegue job di Ford

Francine

Ford:

Non esegue job di Fiona

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

Il Condor pool

- L'installazione di Condor su più macchine forma un pool
- Ogni macchina è configurata con una propria politica, che descrive sotto quali circostanze un job può essere eseguito

condor_status

Comando per avere informazioni sul pool:

```
% condor_status
Name          OpSys  Arch   State     Activ LoadAv Mem   ActvtyTime
perdita.cs.wi LINUX  INTEL  Unclaimed Idle  0.020   511  0+02:28:42
coral.cs.wisc LINUX  INTEL  Claimed   Busy  0.990   511  0+01:27:21
doc.cs.wisc.e LINUX  INTEL  Unclaimed Idle  0.260   511  0+00:20:04
dsonokwa.cs.w LINUX  INTEL  Owner     Idle  0.810   511  0+00:01:45
ferdinand.cs. LINUX  INTEL  Claimed   Suspe 1.130   511  0+00:00:55
```

I 600 job di Frieda termineranno velocemente!

- Anche gli amici di Frieda possono sottomettere i propri job al condor Pool di Frieda!

- Tutti sono felici!!!

Un primo esercizio:

un programma Java

Scelta dell'universo

- Serial Jobs
  - Vanilla
  - Standard
  - Grid
  - Scheduler
  - Local
  - Java
- Parallel Jobs
  - MPI
  - PVM
  - Parallel

Java

- I programmi Java richiedono una Java Virtual Machine (JVM)
- L'ambiente di esecuzione dei Job Java supporta
  - librerie (implementazioni delle classi)
  - I/O

Condor e Java

- L'universo Java fornisce ai programmi java un corretto ambiente di esecuzione
- Condor ha traccia della locazione e della versione della JVM di ogni calcolatore del Pool
- I programmi Java, per la loro esecuzione, vengono indirizzati verso i calcolatori del pool che hanno una JVM installata
- Condor può gestire i file jar
- Condor fornisce informazioni dettagliate su un Job JAVA, non solo sul codice di uscita della JVM: il programma viene eseguito in un wrapper Java che consente a condor di notificare le eccezioni

Es di submit description file dell'universo Java

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

Java support

```
% condor_status –java

 Name          JavaVendor  Ver    State     Actv LoadAv Mem
abulafia.cs Sun Microsy 1.5.0_ Claimed   Busy 0.180   503
acme.cs.wis Sun Microsy 1.5.0_ Unclaimed Idle 0.000   503
adelie01.cs Sun Microsy 1.5.0_ Claimed   Busy 0.000  1002
adelie02.cs Sun Microsy 1.5.0_ Claimed   Busy 0.000  1002

. . .
```

secondo esercizio:

un programma MPI

Scelta dell'universo

- Serial Jobs
  - Vanilla
  - Standard
  - Grid
  - Scheduler
  - Local
  - Java
- Parallel Jobs
  - MPI
  - PVM
  - Parallel

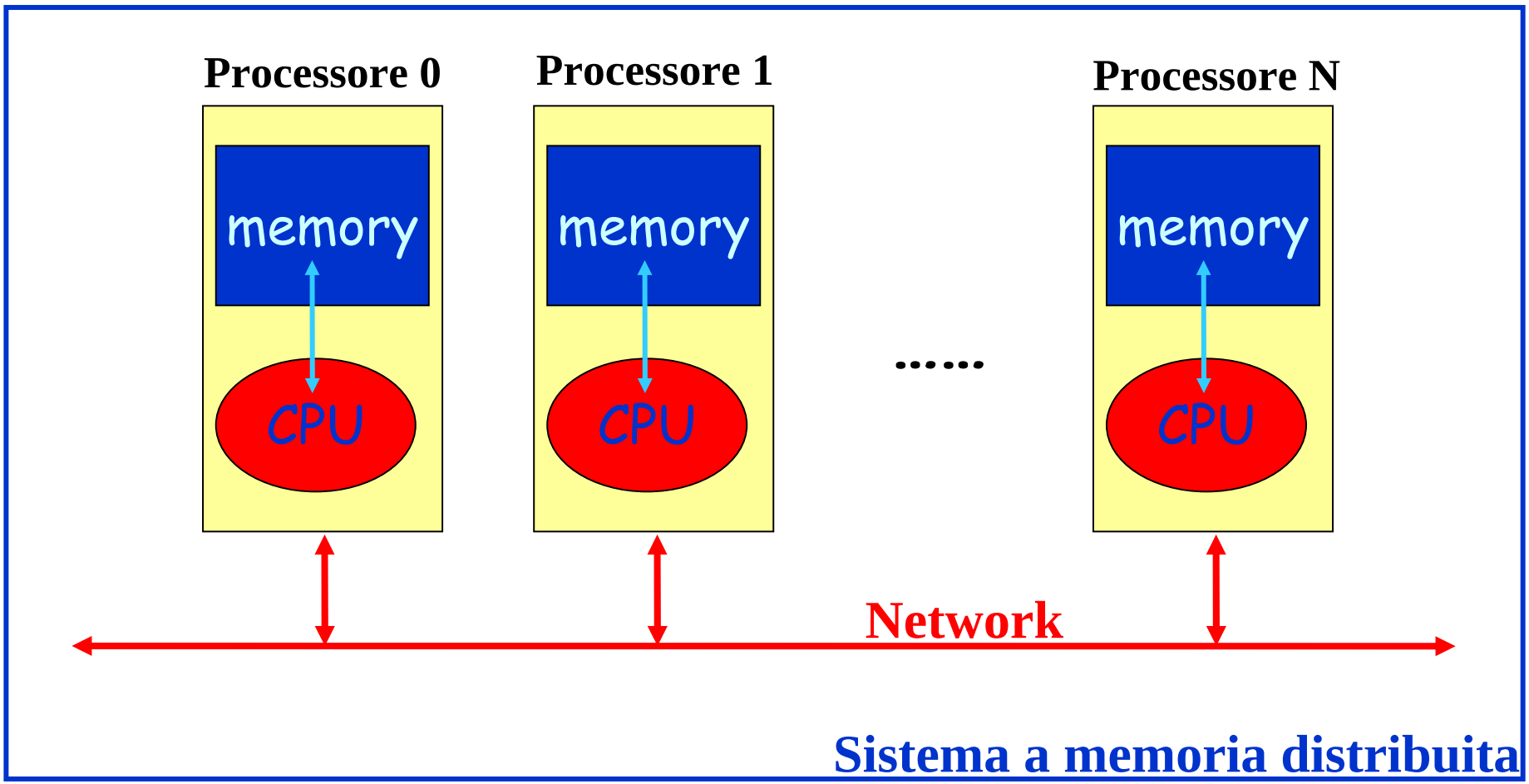
L'universo Parallel

- Per le esecuzioni di programmi paralleli
- Per le esecuzioni di programmi scritti in MPI (Message Passing Interface)

Condor e le applicazioni parallele

- Condor supporta una buona varietà di ambienti di programmazione per il calcolo parallelo
- In particolare, è possibile l'esecuzione su più risorse di un programma scritto in MPI
- Tutte le applicazioni parallele in Condor devono essere sottomesse specificando nel Submision Description File l'universo "parallel"

Si consideri un ambiente di calcolo del tipo…



Ogni processore ha una sua memoria locale alla quale accede direttamente solo la sua CPU. Il trasferimento di dati da un processore ad un altro avviene attraverso una rete.

Il Message Passing

- Il ***Message Passing*** è un modello per la progettazione di algoritmi in un ambiente di Calcolo Parallelo.

- Il modello ***Message Passing*** si basa sull'idea che un insieme di processori può accedere direttamente *solo* alla propria memoria locale, ma può *comunicare* con gli altri inviando e ricevendo messaggi.

- La necessità di standardizzare e rendere portabile il modello ***Message Passing*** ha condotto alla definizione e all'implementazione di un modello standard: Message Passing Interface

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

Caratteristiche di MPI

- **Semplicità di utilizzo**

  Generalmente bastano poche funzioni
  di MPI per scrivere un programma parallelo.

- **Portabilità**

  MPI e' indipendente dall'architettura
  MPI può essere utilizzato sia su sistemi a memoria
  condivisa che su sistemi a memoria distribuita

- **Indipendenza dal linguaggio**

  MPI può essere utilizzato con programmi scritti in
  vari linguaggi: **C, C++, Fortran XX, Java, ...**

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

Esempio di Programma MPI: "hello world"

```
#include <stdio.h>
#include "mpi.h"

int main(int argc, char **argv)
{ int menum,nproc;

  MPI_Int(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&menum);
  MPI_Comm_size(MPI_COMM_WORLD,&nproc);

  printf("hello world I'm %d on %d processors \n"
         ,menun,nproc);

  MPI_Finalize();
  return 0;
}
```

Compilazione (ed esecuzione) di un Programma MPI

```
mpicc –o hello hello.c
```

(Per eseguire il programma ad es. su 8 macchine:

```
mpirun -np 8 hello
```
)

Es di Submit Description File dell'Universo Parallel

```
Universe = parallel
Machine_count = 8
Executable = cpi
Output = cpi.out.$(NODE)
Errore = cpi.err.$(NODE)
Log = cpi.log
Queue
```

Numero di macchine da utilizzare

Ogni processo ha il proprio file per lo standard output ed Error, mentre c'e' *un solo* file di log per il monitoraggio del job parallelo

Vantaggi nell'utilizzare Condor per programmi MPI

- I processi delle applicazioni MPI vengono eseguiti su risorse dedicate
- Pertanto, l'applicazione non verrà mai interrotta né sospesa fino al suo termine (questo accade SOLO per le applicazioni MPI)