



A.D. 1308  
**unipg**  
UNIVERSITÀ DEGLI STUDI  
DI PERUGIA

---

# Relazione Laboratorio HPC

*High Throughput Cluster*

*Esercitazione n°2*

---

Studente: Ludovico Guercio

Matricola: 340036

DIPARTIMENTO DI MATEMATICA E INFORMATICA,  
UNIVERSITÀ DEGLI STUDI DI PERUGIA

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Obbiettivi</b>	<b>1</b>
<b>3</b>	<b>Configurazioni Iniziali</b>	<b>1</b>
3.1	Settings delle Macchine . . . . .	3
3.2	Aggiornamenti e Configurazioni di Rete . . . . .	3
<b>4</b>	<b>Realizzazione del Cluster</b>	<b>4</b>
4.1	Installazione di HTCondor . . . . .	4
4.2	Configurazioni dei Nodi Master e Slave . . . . .	5
4.3	Avvio di HTCondor . . . . .	6
4.4	Configurazione ed Esecuzione di Jobs . . . . .	7
<b>5</b>	<b>Policy per l'esecuzione di Jobs</b>	<b>8</b>
<b>6</b>	<b>Ulteriori Test</b>	<b>10</b>

# 1 Introduzione

La relazione tratta la seconda esercitazione di laboratorio in cui si è stato assegnato il compito di realizzare un cluster ad alta produttività: questi tipi di cluster sono formati da un insieme di nodi capaci di distribuirsi il carico di lavoro in maniera automatizzata, così permettendo l'esecuzione di applicazioni di calcolo che richiedono elevati sforzi a livello di CPU. Ciò è stata guidata tramite la spiegazione degli strumenti e software da utilizzare da parte del prof. Gervasi e il dott. Damiano Perri. Il mio operato, a causa dell'emergenza covid, non è stato svolto fisicamente il laboratorio.

## 2 Obbiettivi

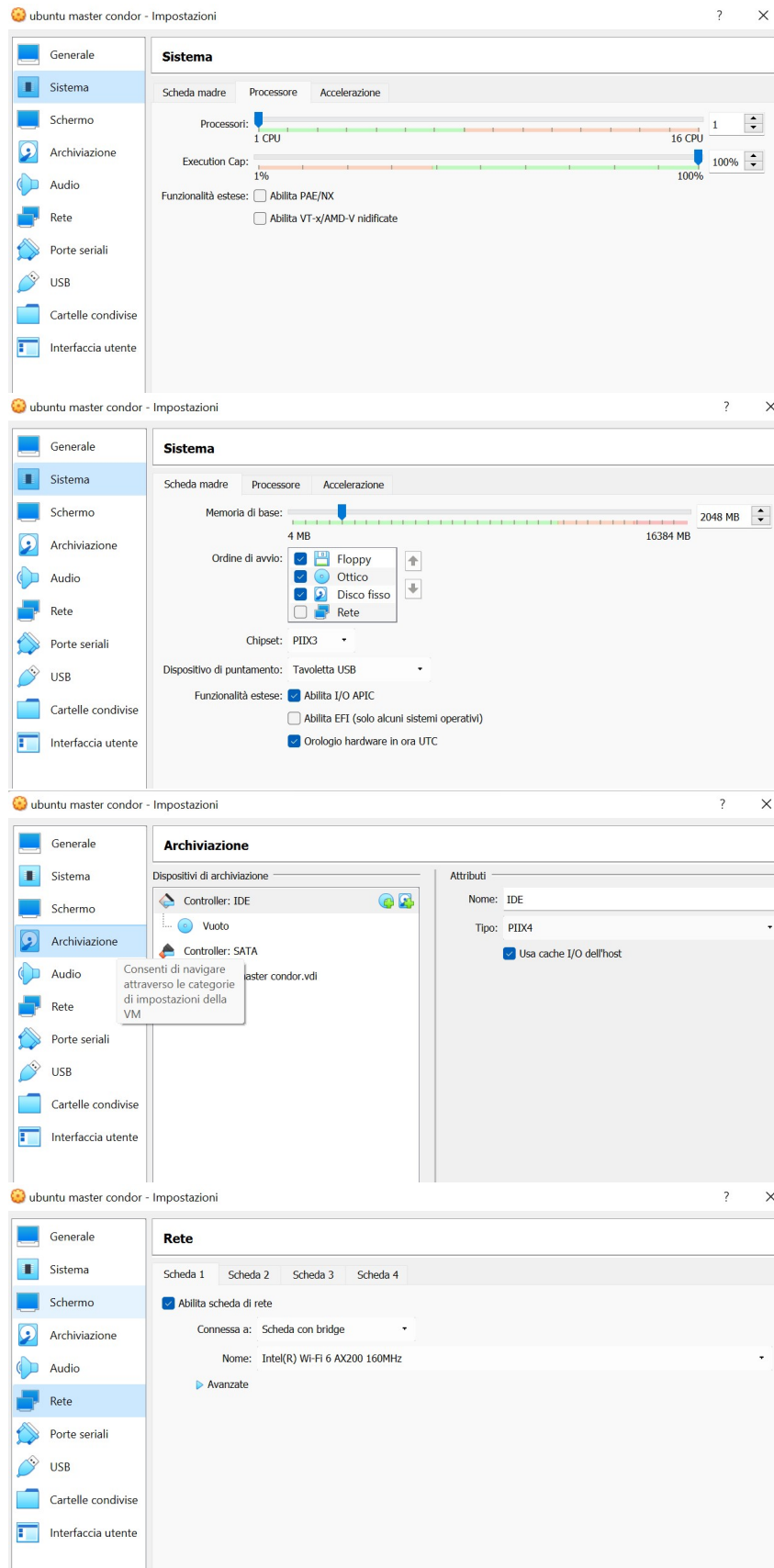
L'obiettivo di questa esercitazione è implementare un cluster ad alta produttività. La realizzazione viene fatta in modo "semplificato" sia per scopo didattico, sia per le configurazioni hardware e software che si ha a disposizione. Il cluster deve essere formato da almeno 2 nodi (sono consigliati 3 nodi), in questi nodi deve essere installato e configurato il software HTCondor capace di fare management e monitoring dei lavori (jobs) e dei nodi nel cluster. Inoltre, bisogna configurare delle opportune policy per l'esecuzione dei jobs in modo da consentire l'esecuzione, lo stop e la ripresa di quest'ultimi in maniera automatica. Infine affinché queste configurazioni siano funzionanti, si provano l'invio e l'esecuzione dei jobs analizzando anche cosa succede se si utilizzano più core nelle macchine.

## 3 Configurazioni Iniziali

L'esercitazione è stata eseguita nel mio laptop con sistema operativo Windows 10; come programma di virtualizzazione delle macchine è stato utilizzato VirtualBox. Il carico da supportare dell'esercitazione non è stato un problema per la mia macchina avendo a disposizione 16 GB di RAM e un processore di nuova generazione.

La mia implementazione del cluster è stata fatta tramite 3 macchine virtuali Ubuntu Server: una macchina lavorerà come nodo **Master**, mentre le altre due lavoreranno come nodi **Slave** dedicate all'esecuzione dei jobs.

Dopo aver scaricato l'iso ufficiale, è stata creata una macchina con le seguenti caratteristiche:



La macchina, come riportato dagli screenshots, ha le seguenti caratteristiche: 2 GB di RAM, 1 Core e 10 GB di memoria del disco e scheda di rete di tipo "bridge".  
Le configurazioni delle macchine saranno tutte uguali, quindi per comodità, si utilizza

la funzione "Clona" di VirtualBox per creare le altre macchine; poi si procede all'avvio delle macchine per eseguire la loro installazione.

### 3.1 Settings delle Macchine

Con il primo avvio delle macchine si procede con l'installazione di esse. Con Ubuntu Server è molto semplice. Ho lasciato le impostazioni predefinite iniziali consigliate per l'installazione, come quelle di formattazione del disco e impostazione del DHCP.

Per ogni macchina poi si settano i vari nomi utenti, nome dei server e password.

Durante l'installazione ho fin da subito installato servizi utili così da non installarli successivamente, tra cui `net-tools`, `openSSH` e `stress-ng` (utile in caso testing con aumento dello sforzo delle CPU).

### 3.2 Aggiornamenti e Configurazioni di Rete

Finite le installazioni di S.O. delle macchine, si procede con il riavvio e si aggiornano i pacchetti per evitare possibili incompatibilità con tools e software non aggiornati.

```
sudo apt update
sudo apt upgrade
sudo apt install net-tools
sudo apt install iptables
```

Per mantenere sempre una comunicazione tra le macchine, è consigliato configurare gli IP statici così evitando un riassegnamento diverso degli indirizzi con il DHCP.

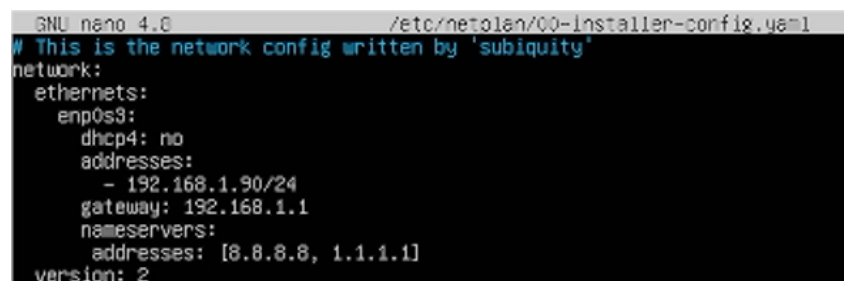
Per settare gli indirizzi statici in Ubuntu Server, bisogna assicurarsi che le impostazioni di Cloud non gestiscano le impostazioni di rete.

Per farlo apro il file di cloud init in

`/etc/cloud/cloud.cfg.d/subiquity-disable-cloudinit-networking.cfg` e controllo che sia presente la voce:

```
"network: {config: disabled}"
```

Nel mio caso il file esiste e questa impostazione è corretta, quindi posso subito proseguire con l'assegnazione degli IP delle macchine andando a modificare il file in `/etc/netplan/00-installer-config.yaml` come segue:



```
GNU nano 4.0 /etc/netplan/00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
        - 192.168.1.90/24
      gateway: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1]
  version: 2
```

Quindi avremo le tre macchine con indirizzi:

- 192.168.1.90 per la macchina **master**
- 192.168.1.91 per la macchina **slaveone**
- 192.168.1.150 per la macchina **slavetwo**

Ho anche assegnato un nome alle macchine nel file che si trova in */etc/hosts*: questo file è quello che viene interrogato per primo quando si effettua una richiesta **dns**; con i nomi presenti nel file sarà concessa una connessione all'indirizzo IP definito.

```
galliani@master:~$ cat /etc/hosts
192.168.1.90 master master.condor
192.168.1.91 slaveone slaveone.condor
192.168.1.150 slavetwo slavetwo.condor
galliani@master:~$
```

Infine, è consigliabile modificare la configurazione del firewall per evitare eventuali problemi o warning di comunicazione fra i nodi e Condor. Nel mio caso, Ubuntu Server di default ha il sistema di firewall disabilitato.

```
galliani@mastercondor:~$ sudo ufw status
Status: inactive
galliani@mastercondor:~$
```

Per attivare definitivamente le nuove impostazioni di rete basta semplicemente riavviare i servizi di rete oppure effettuare un semplice reboot.

## 4 Realizzazione del Cluster

Una volta che le macchine sono pronte ed aggiornate, è dunque possibile proseguire per la realizzazione del cluster. A questo punto si esegue l'installazione del software HTCondor e si vanno a configurare i nodi.

### 4.1 Installazione di HTCondor

Condor è l'unico software utilizzato, quindi si procede in tutte le macchine con il seguente comando:

```
sudo apt install htcondor
```

Nell'installazione è anche possibile scegliere di effettuare l'installazione applicando una configurazione di base offerta da Condor. Nel mio caso questa opzione è stata rifiutata, in quanto devo effettuare delle configurazioni probabilmente diverse e determinate dall'esercitazione.

## 4.2 Configurazioni dei Nodi Master e Slave

Per poter configurare bisogna modificare all'interno di ogni macchina il proprio file di configurazione in `/etc/condor/config.d/`. Dato che ho eseguito l'installazione senza configurazioni, in questo percorso non c'è nessun file, dunque bisogna crearlo ed andarlo a riempirlo:

```
sudo nano /etc/condor/config.d/00personal_condor.config
```

Di base abbiamo due configurazioni ben distinte: una per il nodo che fungerà da **Master** e l'altra per i due **Slave**. La distinzione è particolarmente evidente, e si può notare nella definizione dei demoni per Condor nella `DEAMON LIST`: ad esempio nel nodo Master non avremo `STARTD` perchè questo nodo non è addetto all'esecuzione dei jobs; nei nodi Slave non compaiono `SCHEDD` e `NEGOTIATOR` perchè questi nodi non sono responsabili della gestione e matchmaking dei jobs.

Di seguito le prime configurazioni Condor delle macchine:

### Configurazione nodo Master

```
galliani@mastercondor:~$ cat /etc/condor/config.d/00personal_condor.config
CONDOR_HOST = nodo1

COLLECTOR_NAME = $(FULL_HOSTNAME)

NETWORK_INTERFACE = 192.168.1.90

HOSTALLOW_WRITE = *.condor

DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD

START = True

SUSPEND = False

CONTINUE = True

PREEMPT = False

KILL = False
galliani@mastercondor:~$ _
```

### Configurazione nodi Slave

```
GNU nano 4.0 /etc/condor/config.d/00personal_condor.config
CONDOR_HOST = nodo1

COLLECTOR_NAME = $(FULL_HOSTNAME)

NETWORK_INTERFACE = 192.168.1.91

HOSTALLOW_WRITE = *.condor

DAEMON_LIST = MASTER, STARTD
```

Il nodo1 che rappresenta il nodo Master, fungerà per Condor sia da **Central Manager** che da **Submit**.

Il demone **MASTER** deve essere sempre dichiarato in tutte le macchine, indipendentemente dalle funzioni che ogni singola macchina poi dovrà svolgere; esso è responsabile dell'avvio di tutti gli altri demoni: se un demone ha un problema, Master lo ferma e lo riavvia.

Altri dettagli sui parametri di configurazione:

- **HOSTALLOW WRITE**: permette l'autorizzazione a tutti host, nel mio caso sotto il dominio condor

- **DAEMON LIST**: indica quali demoni caratterizzano la macchina
- **START**: boolean che indica lo stato di una macchina per eseguire un job. Se è True la macchina è pronta per caricarsi il job
- **SUSPEND**: boolean che se è True, indica lo stato di sospensione di un job
- **CONTINUE**: boolean che se è True, indica la ripresa di un job che era stato sospeso
- **PREEMPT**: boolean che se è True, crea un checkpoint e ri-accoda il job per essere assegnato ad un altro slot(core)
- **KILL**: boolean che indica l'interruzione e fine immediata di un job senza tener conto di possibili esecuzioni, interruzioni o checkpoint

Le variabili **START**, **SUSPEND**, **CONTINUE**, **PREEMPT**, **KILL** sono settate, come descritto nel manuale guida, di default; sono adatte per un nodo Master.  
(<https://htcondor.readthedocs.io/en/latest/admin-manual/policy-configuration.html>)

### 4.3 Avvio di HTCondor

Una volta completate le configurazioni delle macchine per Condor, posso avviare il software tramite i comandi:

```
sudo systemctl enable condor.service
sudo systemctl start condor.service
```

```
galliani@mastercondor:~$ condor_status
Name      OpSys      Arch      State      Activity LoadAv Mem      ActvtyTime
slaveone   LINUX      X86_64    Unclaimed Idle        0.130 1983  0+00:00:03

      Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
      X86_64/LINUX      1      0      0      1      0      0      0      0
      Total      1      0      0      1      0      0      0      0
galliani@mastercondor:~$

galliani@slaveone:~$ sudo systemctl status condor.service
[sudo] password for galliani:
• condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/lib/systemd/system/condor.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2022-02-17 11:54:27 UTC; 12min ago
     Main PID: 669 (condor_master)
    Status: "All daemons are responding"
       Tasks: 4 (limit: 2274)
      Memory: 22.3M
    CGroup: /system.slice/condor.service
            └─669 /usr/sbin/condor_master -f
              └─805 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000
                └─807 condor_shared_port -f
                  └─905 condor_startd -f

Feb 17 11:54:27 slaveone systemd[1]: Started Condor Distributed High-Throughput-Computing.
Feb 17 11:54:28 slaveone htcondor[736]: Not changing GLOBAL_MAX_FDS (/proc/sys/fs/file-max): new val
Feb 17 11:54:28 slaveone htcondor[768]: Not changing TCP_LISTEN_QUEUE (/proc/sys/net/core/somaxconn)
Feb 17 11:54:28 slaveone htcondor[774]: Not changing ROOT_MAXKEYS (/proc/sys/kernel/keys/root_maxkey
Feb 17 11:54:28 slaveone htcondor[778]: Not changing ROOT_MAXKEYS_BYTES (/proc/sys/kernel/keys/root
Feb 17 11:54:28 slaveone htcondor[782]: Changing FS_CACHE_DIRTY_BYTES (/proc/sys/vm/dirty_bytes) fr
Feb 17 11:54:28 slaveone htcondor[785]: Changing MAX_RECEIVE_BUFFER (/proc/sys/net/core/rmem_max) fr
```



```

galliani@slavetwo:~$ sudo systemctl status condor.service
[sudo] password for galliani:
condor.service - Condor Distributed High-Throughput-Computing
   Loaded: loaded (/lib/systemd/system/condor.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2022-02-17 11:54:23 UTC; 13min ago
     Main PID: 664 (condor_master)
        Status: "Problems: STARTD=RESTART in 193s"
          Tasks: 3 (limit: 2274)
        Memory: 21.1M
      CGroup: /system.slice/condor.service
              └─ 664 /usr/sbin/condor_master -f
                 └─ 797 condor_procd -A /var/run/condor/procd_pipe -L /var/log/condor/ProcLog -R 1000000
                    └─ 798 condor_shared_port -f

Feb 17 11:54:23 slavetwo systemd[1]: Started Condor Distributed High-Throughput-Computing.
Feb 17 11:54:23 slavetwo htcondor[761]: Not changing TCP_LISTEN_QUEUE (/proc/sys/net/core/somaxconn)
Feb 17 11:54:24 slavetwo htcondor[764]: Not changing ROOT_MAXKEYS (/proc/sys/kernel/keys/root_maxke
Feb 17 11:54:24 slavetwo htcondor[768]: Not changing ROOT_MAXKEYS_BYTES (/proc/sys/kernel/keys/root
Feb 17 11:54:24 slavetwo htcondor[774]: Changing FS_CACHE_DIRTY_BYTES (/proc/sys/vm/dirty_bytes) fr
Feb 17 11:54:24 slavetwo htcondor[777]: Changing MAX_RECEIVE_BUFFER (/proc/sys/net/core/rmem_max) fr

```

Durante il controllo delle configurazioni si verifica un problema: il terzo nodo (ossia la seconda macchina slave) non è stata riconosciuta per il pool per Condor. Dai log e dagli output ho cercato di trovare delle soluzioni al problema nel manuale e online ma non ho trovato situazioni precisamente di problemi simili. Nel tentativo di far funzionare tutti i tre nodi, ho cercato di riconfigurare i file per Condor e anche riconfigurare le impostazioni di rete, il tutto non ha portato alla risoluzione.

Dunque ho deciso di proseguire l'esercitazione con il solo utilizzo di una macchina "worker" sufficiente per testare il funzionamento del cluster e delle policy successivamente implementate.

## 4.4 Configurazione ed Esecuzione di Jobs

Per testare il semplice funzionamento bisogna inviare un job alla rete: è possibile sotto-mettere qualsiasi tipo di programma che sia riconoscibile come un file eseguibile; quindi è libera scelta scrivere o prendere un qualsiasi tipo di programma come in python o C. Per la mia implementazione di un Job ho seguito una ottima e semplice guida ufficiale: <https://www.physics.wisc.edu/computing/condor/>

- Ho creato un file eseguibile in C chiamato `job.c`
- Ho compilato il file ed assegnato i poteri di esecuzione con il comando `chmod +x`
- il programma svolge una semplice moltiplicazione eseguita dopo un tempo di "sleep"

```

GNU nano 4.0                                job.c
#include <stdio.h>
main(int argc, char **argv)
{
    int sleeptime;
    int input;
    int failure;
    if (argc != 3) {
        printf("Usage: simple <sleep-time> <integer> \n");
        failure = 1;
    } else {
        sleeptime = atoi(argv[1]);
        input = atoi(argv[2]);
        printf("Thinking really hard for %d seconds...\n", sleeptime); sleep(sleeptime);
        printf("We calculated: %d\n", input * 2); failure = 0;
    }
    return failure;
}

```

Eseguendolo localmente il programma funziona correttamente. Ora deve essere reso un "job" effettivo per HTCondor: quindi è necessario creare un file di **submit** da mandare al Master, in modo poi da assegnare il job al nodo slave e farlo eseguire.

```
galliani@mastercondor:~$ cat submit
Universe = vanilla
Executable = job
Arguments = 90 10
Log = job.log
Output = job.$(Process).out
Error = job.$(Process).error
Queue
galliani@mastercondor:~$
```

Nel file viene dichiarato quale file deve essere **Executable**, ossia il file **job**, i parametri di input **Arguments**, che sono il tempo di sleep e un int per eseguire la moltiplicazione. Utile anche inserire i parametri per salvare eventuali **Log**, **Output**, **Error**. Il parametro **Queue** indica il numero di volte per il cui il job venga eseguito, ossia la coda dei job. Per il primo semplice test avremo un solo job che sarà assegnato all'unico core disponibile della macchina slave. L'invio del job è fatto tramite il comando:

```
condor_submit <file_submit>
```

```
galliani@mastercondor:~$ condor_submit submit
Submitting job(s).
1 job(s) submitted to cluster 4.
galliani@mastercondor:~$ condor_status
Name OpSys Arch State Activity LoadAv Mem ActvtyTime
slaveone LINUX X86_64 Claimed Busy 0.000 1983 0+00:00:02

Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
X86_64/LINUX 1 0 1 0 0 0 0 0 0
Total 1 0 1 0 0 0 0 0 0
galliani@mastercondor:~$
```

L'esecuzione del job funziona correttamente, il Master è riuscito ad assegnare e fare eseguire il job al core del nodo Slave: infatti il nodo **slaveone** passa dallo stato **Unclaimed/Idle** a **Claimed/Busy**. Naturalmente nel caso si mandassero più job, ad esempio con una **Queue** di 4 job, Condor avendo a disposizione solo un core della macchina **slaveone** non può distribuire i lavori a qualcun'altro; dunque, i job saranno in coda sempre per lo stesso slot.

## 5 Policy per l'esecuzione di Jobs

Nell'esercitazione è anche richiesto, per raggiungere un migliore risultato, di definire delle opportune policy per l'esecuzione dei jobs all'interno del cluster. Con l'utilizzo delle policy si può vedere il vero funzionamento per cui si utilizza il software Condor. La policy all'interno di questo sistema deve far sì che l'esecuzione di un job o più jobs termina non

appena viene rilevata attività da tastiera oppure quando la CPU viene stressata oltre il 60 %. La policy poi deve far sì che l'esecuzione dei jobs avvenga soltanto nelle macchine in stato di IDLE e che i jobs vengano stoppati e ripresi in modo automatico.

Dunque, sono andato a modificare i file di configurazione `00personal_condor.config` della macchina Slave aggiungendo quindi i parametri necessari:

```
GNU nano 4.8 /etc/condor/config.d/00personal_condor.config Modified
CONDOR_HOST = master

COLLECTOR_NAME = $(FULL_HOSTNAME)

NETWORK_INTERFACE = 192.168.1.91

HOSTALLOW_WRITE = *.condor

DAEMON_LIST = MASTER, STARTD

#macro per policy
MINUTE = 60
StateTimer = (time() - EnteredCurrentState)
ActivityTimer = (time() - EnteredCurrentActivity)
ActivationTimer = (time() - JobStart)

KeyboardBusy = KeyboardIdle < 30
HighLoad = 0.6

NonCondorLoadAvg = (LoadAvg - CondorLoadAvg)
BackgroundLoad = 0.3
CPUIIdle = $(NonCondorLoadAvg) <= $(BackgroundLoad)
CPUBusy = $(NonCondorLoadAvg) >= $(HighLoad)
MachineBusy = $(CPUBusy) || $(KeyboardBusy)

WANT_SUSPEND = TRUE
WANT_VACATE = TRUE
START = $(CPUIIdle)
SUSPEND = $(MachineBusy)
CONTINUE = ( $(CPUIIdle) && $(ActivityTimer) > $(MINUTE) )
PREEMPT = ((Activity == "Suspended") && $(ActivityTimer) > 2 * $(MINUTE))
KILL = $(ActivityTimer) > 4 * $(MINUTE)
```

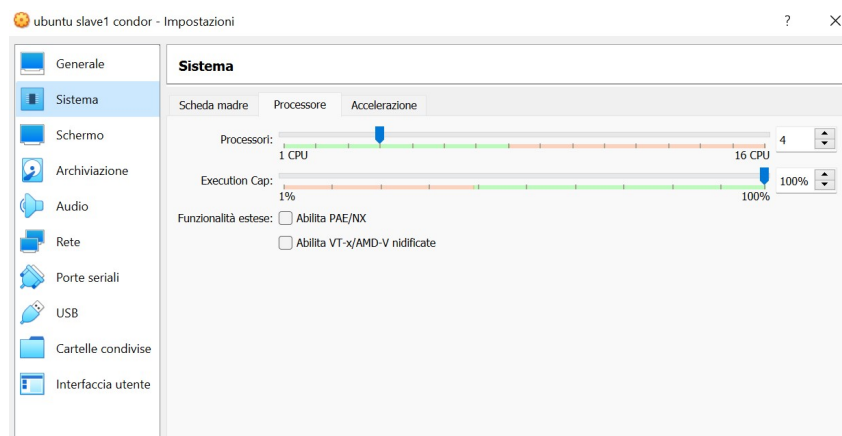
Nuovi parametri:

- **ActivityTimer**: quantità di tempo in secondi nell'attività corrente
- **KeyboardBusy**: boolean che indica se c'è presenza di attività da tastiera
- **KeyboardIdle**: boolean che indica la non presenza di attività da tastiera
- **CPUBusy**: boolean che indica se la CPU è occupata per un lavoro
- **CPUIIdle**: boolean che indica se la CPU è in stato IDLE
- **MachineBusy**: indica la condizione per cui la CPU o la tastiera sono in attività
- **HighLoad**: parametro che indica il valore massimo di sforzo settato per la CPU; se **NonCondorLoadAvg** va oltre, la CPU è considerata troppo occupata e dovrebbe iniziare l'interruzione del job.
- **NonCondorLoadAvg**: la differenza tra il carico di sistema e il carico Condor (il carico generato da tutto tranne da Condor).
- **Configurazioni Policy**:
  - **WANT\_SUSPEND**: boolean che in base al suo valore indica a Condor come valutare la variabile **CONTINUE**; se True deve valutarla
  - **WANT\_VACATE**: boolean che in base al suo valore indica a Condor come valutare la variabile **PREEMPT**; se True deve valutarla

- **START**: quando la CPU è IDLE, ossia è sotto il 30% di sforzo, la macchina si può caricare un job
- **SUSPEND**: quando la macchina è occupata (`MachineBusy` è `True`), avviene la sospensione di un job
- **CONTINUE**: quando la CPU è IDLE, ossia sotto il 30% di sforzo, e c'è sospensione da più di un minuto, riprende l'esecuzione del job sospeso
- **PREEMPT**: quando ci si trova nello stato di sospensione (`Suspended`) da più di 2 minuti, effettua la preemption del job
- **KILL**: quando `ActivityTimer` supera la soglia di 4 minuti, avviene l'interruzione ed eliminazione immediata di un job

## 6 Ulteriori Test

Con la configurazione delle policy, il cluster deve essere in grado di saper gestire le interruzioni e il ri-assegnamento dei lavori nelle CPU. Utile a questo punto aumentare il numero di core della macchina `slaveone`: questo passaggio è semplicemente fatto direttamente dalle impostazioni della macchina in Virtualbox:



Ricollegando nella macchina, tramite il comando `condor_status` si vedono facilmente i nuovi slot della CPU che sono stati riconosciuti:

```
galliani@mastercondor:~$ condor_status
Name           OpSys      Arch      State      Activity LoadAv Mem      ActvtyTime
slot1@slaveone LINUX      X86_64    Unclaimed Idle       0.170  495  0+00:00:03
slot2@slaveone LINUX      X86_64    Unclaimed Idle       0.000  495  0+00:00:24
slot3@slaveone LINUX      X86_64    Unclaimed Idle       0.000  495  0+00:00:24
slot4@slaveone LINUX      X86_64    Unclaimed Idle       0.000  495  0+00:00:24
```

Per verificare il funzionamento della sospensione di un job, ho utilizzato il tool `stress-ng` (installato nella fase di installazione del sistema operativo) per aumentare lo sforzo su più core così da portarle sicuramente oltre il 60%, così da soddisfare la condizione di sospensione della policy. Inoltre ho modificato il file di `submit` del job andando ad aumentare il numero dei job con una `Queue = 2` (si può anche modificare il parametro di sleep time aumentandolo

di molto così permettendo a Condor di valutare tutti i parametri.

Avendo a disposizione soltanto il funzionamento di una sola macchina worker, non sarà possibile ad esempio mostrare come la policy può, dopo la sospensione di job a causa di un elevato sforzo della CPU, riassegnare il job ad un altro nodo per poter riprendere l'esecuzione del lavoro.

A questo punto si può sottomettere il job sempre attraverso il comando `condor_submit`

```
galliani@mastercondor:~$ condor_status
Name           OpSys      Arch      State      Activity LoadAv Mem   ActvtyTime
slot1@slaveone LINUX      X86_64    Claimed    Busy       0.000  495  0+00:00:03
slot2@slaveone LINUX      X86_64    Claimed    Busy       0.000  495  0+00:00:03
slot3@slaveone LINUX      X86_64    Unclaimed  Idle       0.000  495  0+00:00:24
slot4@slaveone LINUX      X86_64    Unclaimed  Idle       0.000  495  0+00:00:24

Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
X86_64/LINUX  4      0      2      2      0      0      0      0
Total        4      0      2      2      0      0      0      0
galliani@mastercondor:~$
```

I job sono stati presi correttamente dai primi due slot della macchina; ora per mandarli nello stato di "Suspend", utilizzo il comando nella macchina slave:

```
stress-ng --cpu 2
```

I processi di `stress-ng` hanno portato le CPU al massimo, quindi andando a controllare lo status delle CPU in Condor si deve verificare la sospensione dei job: quindi ora la macchina `slaveone` deve passare anche nello stato di `Owner/Idle`.

```
galliani@master:~$ condor_status
Name           OpSys      Arch      State      Activity LoadAv Mem   ActvtyTime
slot1@slaveone LINUX      X86_64    Unclaimed  Idle       0.000  495  0+00:00:03
slot2@slaveone LINUX      X86_64    Unclaimed  Idle       0.000  495  0+00:00:03
slot3@slaveone LINUX      X86_64    Owner      Idle       0.310  495  0+00:00:04
slot4@slaveone LINUX      X86_64    Owner      Idle       0.320  495  0+00:00:03

Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
X86_64/LINUX  4      2      0      2      0      0      0      0
Total        4      2      0      2      0      0      0      0
galliani@master:~$
```

Dallo screenshot si è catturato dal momento in cui si verifica anche il rilascio del job passano allo stato di `Idle`.

Ora in questo stato, avendo a disposizione solo questa macchina con liberi i primi due slot CPU, dovrebbe avvenire di nuovo il riassegnamento dei due job:

```
galliani@master:~$ condor_status
Name           OpSys      Arch      State      Activity LoadAv Mem   ActvtyTime
slot1@slaveone LINUX      X86_64    Claimed    Busy       0.000  495  0+00:00:03
slot2@slaveone LINUX      X86_64    Claimed    Busy       0.000  495  0+00:00:03
slot3@slaveone LINUX      X86_64    Owner      Idle       1.000  495  0+00:04:17
slot4@slaveone LINUX      X86_64    Owner      Idle       1.000  495  0+00:03:23

Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
X86_64/LINUX  4      2      2      0      0      0      0      0
Total        4      2      2      0      0      0      0      0
```

Avendo di nuovo i job in esecuzione, posso subito provare se effettuando qualsiasi attività da tastiera nella macchina `slaveone` i job si sospendono:

```

galliani@master:~$ condor_status
Name           OpSys      Arch    State      Activity LoadAv Mem    ActvtyTime
slot1@slaveone LINUX      X86_64 Claimed   Busy      0.040 495 0+00:02:41
slot2@slaveone LINUX      X86_64 Claimed   Busy      0.000 495 0+00:02:41
slot3@slaveone LINUX      X86_64 Owner    Idle      1.000 495 0+00:09:17
slot4@slaveone LINUX      X86_64 Owner    Idle      1.000 495 0+00:08:23

      Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
      X86_64/LINUX      4      2      2          0          0          0          0          0

      Total      4      2      2          0          0          0          0          0
galliani@master:~$ condor_status
Name           OpSys      Arch    State      Activity LoadAv Mem    ActvtyTime
slot1@slaveone LINUX      X86_64 Claimed   Suspended 0.010 495 0+00:00:03
slot2@slaveone LINUX      X86_64 Claimed   Suspended 0.000 495 0+00:00:03
slot3@slaveone LINUX      X86_64 Owner    Idle      1.000 495 0+00:09:17
slot4@slaveone LINUX      X86_64 Owner    Idle      1.000 495 0+00:08:23

      Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
      X86_64/LINUX      4      2      2          0          0          0          0          0

      Total      4      2      2          0          0          0          0          0

```

Attendendo sempre un po' di tempo, i job devono essere ripresi per la loro esecuzione e la loro terminazione:

```

galliani@master:~$ condor_status
Name           OpSys      Arch    State      Activity LoadAv Mem    ActvtyTime
slot1@slaveone LINUX      X86_64 Claimed   Busy      0.100 495 0+00:00:42
slot2@slaveone LINUX      X86_64 Claimed   Busy      0.000 495 0+00:00:42
slot3@slaveone LINUX      X86_64 Owner    Idle      1.000 495 0+00:14:17
slot4@slaveone LINUX      X86_64 Owner    Idle      1.000 495 0+00:13:23

      Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
      X86_64/LINUX      4      2      2          0          0          0          0          0

      Total      4      2      2          0          0          0          0          0
galliani@master:~$ condor_status
Name           OpSys      Arch    State      Activity LoadAv Mem    ActvtyTime
slot1@slaveone LINUX      X86_64 Unclaimed Idle      0.000 495 0+00:00:40
slot2@slaveone LINUX      X86_64 Unclaimed Idle      0.000 495 0+00:00:41
slot3@slaveone LINUX      X86_64 Owner    Idle      1.000 495 0+00:19:17
slot4@slaveone LINUX      X86_64 Owner    Idle      1.000 495 0+00:18:23

      Total Owner Claimed Unclaimed Matched Preempting Backfill Drain
      X86_64/LINUX      4      2      0          2          0          0          0          0

      Total      4      2      0          2          0          0          0          0

```