Guida introduttiva e regole per l'uso dei cluster HPC

- Staff HPC@Polito -

Questa guida è rivolta agli utenti dei nostri cluster per rispondere alle frequenti domande sull'uso dei sistemi HPC
Per maggiori informazioni contattare lo Staff alla seguente e-mail
hpc.dauin@polito.it

Indice

Destinatari

La corrente guida illustra l'utilizzo dei sistemi HPC@POLITO basati su scheduler SLURM. Il documento può essere utilizzato per tutti i cluster, è sufficiente sostituire a {login-node} il nome del nodo di login per il cluster che si intende usare:

[CASPER] *casperlogin.polito.it*

[HACTAR] *hactarlogin.polito.it*

[LEGION] *legionlogin.polito.it*

Gentlemen's agreement

- Utilizzando i nostri sistemi HPC@POLITO per scopi di ricerca, l'utente autorizza automaticamente lo Staff HPC@POLITO a pubblicare i propri dati personali (nome, cognome, gruppo di ricerca) e i dati associati alla ricerca sul sito web (hpc.polito.it) e in tutte le altre pubblicazioni cartacee divulgate da HPC@POLITO (report annuali, presentazioni, etc.), nonche su qualsiasi altro supporto.
- Utilizzando i nostri sistemi HPC@POLITO per scopi di ricerca, l'utente accetta di citare il centro di calcolo HPC@POLITO su tutti i propri articoli scientifici in paper, conferenze, libri o altro tipo di supporto. Suggeriamo la seguente citazione:

  "Risorse di calcolo fornite HPC@POLITO, progetto di Academic Computing del Dipartimento di Automatica e Informatica presso il Politecnico di Torino (http://www.hpc.polito.it)".

  O la seguente versione più breve:

  "Risorse di calcolo fornite da HPC@POLITO (http://www.hpc.polito.it)".

Regole per l'uso dei cluster

- NON devono mai essere avviati calcoli, simulazioni ecc. direttamente da linea di comando rischiando così di rendere inutilizzabile il nodo di login o altre risorse condivise (pena il blocco dell'account e l'interruzione dei task). Passare SEMPRE attraverso le code di esecuzione dello scheduler (anche in caso di compilazione).
- NON è consentito qualsiasi uso delle risorse che esuli dall'attività didattica e/o di ricerca.
- Ogni utente è responsabile per qualunque attività svolta o riconducibile allo stesso, è pertanto scoraggiato l'account sharing, consentito solo se segnalato per tempo allo Staff di HPC.

  In generale, NON è permessa la condivisione a terzi di dati sensibili (nome utente, password) che l'utente si impegna a custodire adeguatamente.
- Come norma generale, NON è consentito fare nulla che NON sia espressamente consentito.

Il presente documento integra il regolamento del centro di calcolo HPC@POLITO con informazioni di carattere tecnico e pratico sul funzionamento dello stesso.
Versioni aggiornate del regolamento e del relativo manuale d'uso possono essere scaricate dal sito web del progetto: *http://hpc.polito.it*.
In particolare si ricorda che tutti gli utenti sono tenuti al rispetto del Regolamento.

HPC@POLITO Academic Computing Center

Il progetto HPC del Politecnico di Torino è un centro di Academic Computing, ovvero fornisce le risorse di calcolo e il supporto tecnico per attività di ricerca accademica e didattica facenti uso dei sistemi del centro.
Il progetto HPC è ufficialmente gestito dal LABINF (Laboratorio Didattico di Informatica Avanzata) con la supervisione del DAUIN (Dipartimento di Automatica e Informatica) su conferimento del consiglio di amministrazione.

Casper - Specifiche tecniche

Architettura	Cluster Linux Infiniband-DDR MIMD Distributed Shared-Memory
Interconnessione nodi	Infiniband DDR 20 Gb/s
Rete di servizio	Ethernet 1 Gb/s
Modello CPU	2x Opteron 6276/6376 (Bulldozer) 2.3 GHz (turbo 3.0 GHz) 16 cores
Performance	4.360 TFLOPS
Core computazionali	512
Numbero di nodi	16
Memoria RAM totale	2.0 TB DDR3 REGISTERED ECC
OS	Centos 7.6 - OpenHPC 1.3.8.1
Scheduler	SLURM 18.08

Hactar - Specifiche tecniche

Architettura	Cluster Linux Infiniband-QDR MIMD Distributed Shared-Memory
Interconnessione nodi	Infiniband QDR 40 Gb/s
Rete di servizio	Ethernet 1 Gb/s
Modello CPU	2x Xeon E5-2680 v3 2.50 GHz (turbo 3.3 GHz) 12 cores
Nodo GPU	2x Tesla K40 - 12 GB - 2880 cuda cores
Performance	20.1 TFLOPS (giugno 2018)
Core computazionali	696
Numbero di nodi	29
Memoria RAM totale	3.7 TB DDR4 REGISTERED ECC
OS	CentOS 7.6 - OpenHPC 1.3.8.1
Scheduler	SLURM 18.08

Legion - Specifiche tecniche

Architettura	Cluster Linux Infiniband-EDR MIMD Distributed Shared-Memory
Interconnessione nodi	Infiniband EDR 100 Gb/s
Rete di servizio	Ethernet 1 Gb/s
Modello CPU	2x Intel Xeon Scalable Processors Gold 6130 2.10 GHz 16 cores
Nodo GPU	24x nVidia Tesla V100 SXM2 - 32 GB - 5120 cuda cores
Performance	90 TFLOPS (luglio 2020)
Core computazionali	1824
Numbero di nodi	57
Memoria RAM totale	22 TB DDR4 REGISTERED ECC
OS	CentOS 7.6 - OpenHPC 1.3.8.1
Scheduler	SLURM 18.08

Storage - Specifiche tecniche

Home Storage BeeGFS	349 TB, throughput maggiore di 5 GB/s
Rete di interconnessione	Infiniband EDR 100Gb/s (legion), Ethernet 10 Gb/s (hactar & casper)

Scandenza account e policy archiviazione dati

Account

L'account è da considerarsi attivo dal momento in cui si riceve la email di conferma contenente le proprie credenziali di accesso.

L'account scade secondo quanto indicato nel modulo di richesta. Per rinnovare l'account è necessario inviare nuovamente il medesimo modulo inviato nella prima richiesta avendo cura di modificare i campi con le informazioni aggiornate. Per conoscere la data di scadenza del prorio account è possibile utilizzare il comando:

$ accountExpire

Archiviazione dati

Se l'account non viene riattivato nei 12 mesi successivi alla scadenza indicata nel modulo di richiesta, vengono creati due archivi compressi contenenti tutti i file presenti nella home directory e nella propria directory in work.
Gli archivi compressi sono collocati nella stessa home directory per quello relativo alla home e nella propria directory in work per quello relativo a work.
Trascorsi ulteriori 12 mesi dalla creazione degli archivi compressi (totale 2 anni dall'expire dell'account), tutti gli archivi sono rimossi definitivamente dallo storage.

Quote storage

Quota storage a disposizione degli utenti:

- /home ogni utente può utilizzare fino a 2TB di spazio disco

Gli utenti possono richiedere un aumento della quota loro riservata inviandoci una mail e includendo le opportune motivazioni, la richiesta verrà quindi valutata dallo Staff.
Eventuali estensioni alle quote relative allo spazio utilizzabili per i dati degli utenti devono avere una scadenza, la scadenza deve avere una durata il più possibile limitata al periodo di reale necessità, il periodo richiesto per la quota estesa non può superare la data di expire prevista per l'account.
Per gli account che risulteranno overquota al controllo mensile successivo all'expire dello stesso, si procederà all'archiviazione dei dati anticipatamente non appena sarà trascorso il normale grace time concesso (7 giorni).
Se al successivo controllo mensile l'account risulterà ancora overquota anche a valle della creazione anticipata degli archivi compressi, si procederà alla rimozione anticipata dei file di archivio;

Primo accesso

Raccomandiamo caldamente il cambiamento della password al primo log-in usando il seguente comando:

$ passwd

In ogni caso la password ha scadenza annuale, gli utenti possono provvedere a rinnovarla mediante il comando per la modifica.
Non esistono restrizioni sulla lunghezza e sulla composizione delle password d'accesso, tuttavia si consiglia di scegliere una password composta da 8 o più caratteri, contenente numeri e lettere maiuscole e minuscole.
Come accedere al cluster? Dipende da quale sistema operativo è utilizzato sul proprio calcolatore.
Nel caso si disponga di un sistema *Linux, Unix o OSX*, si può utilizzare il client *ssh* da un qualunque terminale tramite il comando:

$ ssh username@{login-node}

specificando username e password incluse nella email di conferma.

Nel caso si disponga di un sistema *Windows* si consiglia l'uso dell'applicativo PuTTY (disponibile all'indirizzo http://www.putty.org) da configurarsi come in figura:

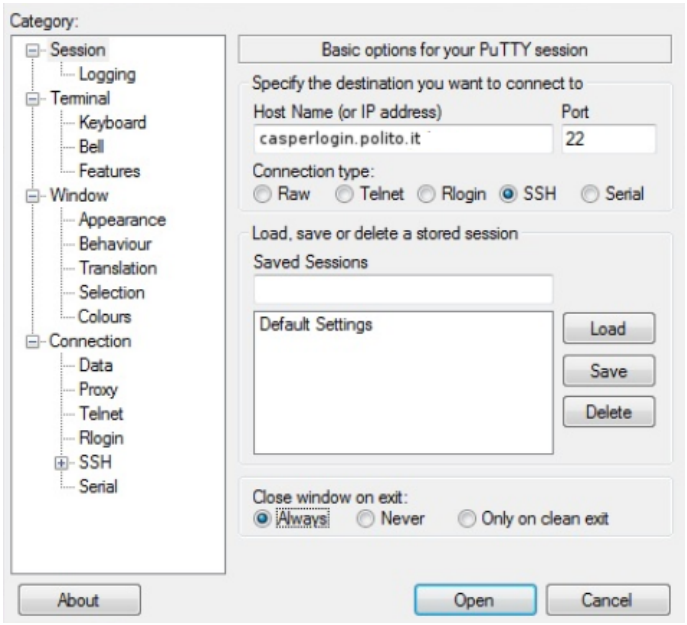

Figura 1: Putty - Configurazione per l'accesso da sistemi Windows

Si ricorda ancora, che non è consentita la pratica dell'account sharing (ovvero la condivisione delle credenziali di accesso tra più persone), inoltre, chi ha fatto richiesta dell'account deve prendersi cura delle proprie credenziali: scegliendo password opportune, non rivelando a terzi dati sensibili e segnalando per tempo allo Staff di HPC l'eventuale compromissione della propria utenza.

Trasferimento file e storage

Trasferimento file

Una volta effettuato l'accesso ci si troverà direttamente nella propria home directory attraverso l'*Home Storage*, accessibile su tutti i cluster da:

/home/username/

qui andranno caricati i dati necessari all'avvio dei task e i dati da salvare al termine di ogni task. L'accesso a questa directory è consentito solo all'utente proprietario che può utilizzare fino a 2TB di spazio disco.
Per trasferire i file dal proprio host all'interno della propria home directory sul cluster è possibile utilizzare il comando *scp*:

$ scp -r /path/to/local/dir/ username@{login-node}:/home/username

Viceversa per copiare un file dal CLUSTER al proprio host usare il seguente comando:

$ scp -r username@{login-node}:/home/username /path/to/local/dir/

Il suo funzionamento è simile al comando *cp* in ambienti Unix.

$ cp SORGENTE DESTINAZIONE

NOTA: si rimanda alla pagina di manuale di *scp* per informazioni dettagliate sul comando e sulle sue opzioni.

Come terza opzione, per trasferire file sul cluster è possibile l'utilizzo del protocollo SFTP.
Ad esempio, è possibile sfruttare un programma come FileZilla, configurato come in figura.

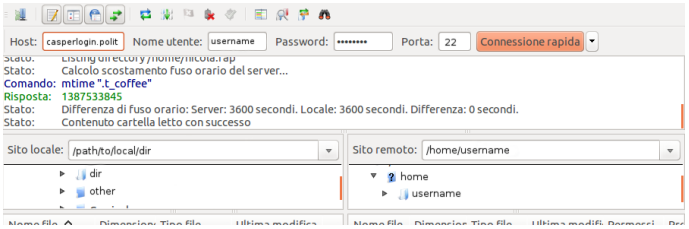

Figura 2: Configurazione FileZilla per il trasferimento file su {login-node}

Home storage BeeGFS

BeeGFS è un file system parallelo, sviluppato e ottimizzato per l'elaborazione ad alte prestazioni, è caratterizzato da un'architettura con metadati distribuiti per motivi di scalabilità e flessibilità.
Lo storage è configurato in modalità Buddy Mirror, questo comporta che tutti i file sono replicati in due copie su server differenti.
Lo storage ha una capacità di 349TB e può raggiungere performance di 1.1GB/s se usato da CASPER e HACTAR, oltre 5 GB/s da LEGION.
Per conoscere lo stato della propria quota usare il comando:

$ get-my-beegfs-quota

..... *Esempio* .....

```
$ get-my-beegfs-quota

Quota information for storage pool Default (ID: 1):

     user/group     ||           size           ||    chunk files
    name     |  id  ||     used     |     hard     ||  used    |  hard
-------------|------||--------------|--------------||----------|----------
  myusername |  1120||     2.20 GiB |     2.00 TiB||       328|  2000000
```

N.B. I dati riportati dal comando riportano lo spazio occupato anche dei file replicati, per conoscere quindi lo spazio occupato è sufficente dividere per due i valori riportati.

Aumento quota

Gli utenti possono richiedere un aumento della quota loro riservata, inviando una mail a *hpc.dauin@polito.it* e includendo le oppurtune motivazioni; la richiesta verrà quindi valutata dallo Staff di HPC@POLITO.

Storage locale ai nodi

Per ogni job è possibile utilizzare temporaneamente lo spazio addizionale presente sui dischi locali dei nodi computazionali.
Tale spazio è accessibile al seguente percorso:

/scratch 1TB

All'interno del job è possibile fare riferimento a una specifica directory temporanea creata automaticamente, sullo storage locale del nodo, all'avvio del job e cancellata al suo termine. La variabile di ambiente $TMPDIR contiene il path alla cartella temporanea del job.

ATTENZIONE:

1. lo spazio disponibile dipende dall'uso della risorsa anche da parte degli altri job
2. l'uso del path */scratch* è deprecato, utilizzare la directory gestita automaticamente e specificata dalla variabile di ambiente $TMPDIR
3. la cartella in $TMPDIR viene eliminata al termine del job, se il vostro programma salva dell'output nella cartella è necessario trasferirlo nello storage HOME

Esemio sbatch con $TMPDIR

[esempio_script_con_tmpdir.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=name.surname@polito.it
#SBATCH --partition=global
#SBATCH --time=hh:mm:ss
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=1024M

# Copia dei file da storage HOME a storage locale del nodo
cp -r /home/myusername/data $TMPDIR/data

myexecutable -in $TMPDIR/data -out $TMPDIR/results
```

```
        # Copia dei risultati qualora generati nella cartella temporanea
        cp -r $TMPDIR/results /home/myusername/results
```

.....

Uso del cluster

I processi eseguibili su un cluster sono processi di tipo "batch": questo significa che non sono interattivi e che la loro esecuzione può essere rimandata nel tempo. Ogni task o job è costituito da uno o più processi che cooperano assieme per raggiungere un determinato risultato.
Un task viene eseguito solo dopo il suo scheduling; per consentirne la schedulazione il job deve essere inserito in una coda, gestita dal cluster, in cui resta in attesa che le risorse necessarie, eventualmente detenute da altri processi, siano disponibili. I nostri cluster utilizzano lo scheduler SLURM per la gestione di tali code di attesa per le risorse condivise.
Le partizioni pubbliche per la sottomissione del job sono:

CASPER:

global partizione di default ad accesso pubblico che include tutti i nodi.; la massima durata di un job è 10 giorni.

HACTAR:

global partizione di default ad accesso pubblico che include tutti i nodi; la massima durata di un job è 10 giorni.

cuda partizione per task facenti uso di GPU, 2 slot GPU per nodo; la massima durata di un job è 10 giorni. E' obbligatorio usare l'opzione *--gres=gpu:*{*N_of_gpu*} (senza parentesi graffe).

LEGION:

global partizione di default ad accesso pubblico che include tutti i nodi; la massima durata di un job è 10 giorni.

cuda partizione per task facenti uso di GPU, 4 slot GPU per nodo; la massima durata di un job è 5 giorni. E' obbligatorio usare l'opzione *--gres=gpu:*{*N_of_gpu*} (senza parentesi graffe).

Controllo e sottomissione di un job

sbatch

Tutti i job devono essere passati allo scheduler del clustrer tramite sottomissione. La sottomissione dei job avviene tramite il comando *sbatch* che ha come argomento il nome completo di uno script contenente tutte le informazioni necessarie.
Lo script file, anche detto *sbatch script*, è composto da una serie di direttive per lo scheduler, seguite da una serie di comandi così come andrebbero digitati sulla command line.

Un esempio di script *sbatch* è il seguente, tutte le linee che iniziano con `#SBATCH` sono opzioni per lo scheduler, e non vengono interpretate dalla shell Linux.

[script.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=name.surname@polito.it
#SBATCH --partition=global
#SBATCH --time=hh:mm:ss
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=1024M
example_task.bin
```

Ora è possibile invocare lo script con:

```
$ sbatch script.sbatch
```

Per creare il file `script.sbatch` è possibile utilizzare il proprio PC e in seguito trasferirlo sul cluster, oppure editarlo direttamente dal cluster.
`NOTE:` i file di testo creati su host DOS/Windows hanno un differente carattere di fine linea rispetto a quelli creati con sistemi Unix-like. DOS usa 'carriage-return' e 'line feed', mentre Unix utilizza semplicemente 'line-feed'. Durante un trasferimento di file tra host Windows e Unix occorre quindi prestare attenzione e assicurarsi che i fine linea siano correttamente tradotti.

Direttive principali:

`--output`	Lo standard output è rediretto al file specificateo, di default sia lo standard output che lo standard error vengono rediretti al medesimo file.
`--error`	Lo standard error è rediretto al file specificato.
`--mail-user`	e-mail a cui inviare le informazioni sul task in esecuzione
`--mail-type`	Eventi scatenanti una notifica via a-mail (*es. ALL*). Valori validi sono: NONE, BEGIN, END, FAIL, REQUEUE, ALL.
`--workdir={directory}`	Esegue il task usando la {directory} specificata come working directory (può essere specificato sia un percorso assoluto che relativo).
`--ntasks-per-node`	Numero di task per nodo, se usato insiema a `--ntasks` quest'ultima direttiva avrà la precedenza.
`--cpus-per-task`	Numero di CPU per task.
`--ntasks`	Numero totale di task per job.
`--nodes`	Numero di nodi da utilizzare.
`--time`	Specifica il limite massimo di run-time, ovvero il tempo necessario al processo per raggiungere il termine della computazione. Tale valore deve essere inferiore a 10 giorni (<240 ore) per task su partizione.
`--mem-per-cpu`	Specifica la memoria minima richiesta per CPU allocata, espressa in megabytes (default=1000).
`--mem`	Specifica la reale memoria richiesta per nodo, espressa in megabytes.
`--partition`	Indica la partizione su cui il job deve essere schedulato (default=global).
`--exclude`	Esclude esplicitamente i nodi specificati dall'insieme di risorse concesse al job.
`--constraint`	I nodi hanno assegnate alcune feature, gli utenti possono specificare quali feature saranno richieste dal loro job utilizzando questa direttiva, ad esempio `--constraint="gpu"`. Le feature disponibili si possono visualizzare nella sezione *Scheduling pool data* nell'output del comando `sjstat` (colonna *Other Traits*), oppure tramite `scontrol show node`.
`--gres=gpu:{N_of_gpu}`	Risorsa generica richiesta per nodo, usato per specificare la richiesta di GPU per nodo.

Per la guida completa fare riferimento a https://slurm.schedmd.com/sbatch.html.

squeue, sinfo, sjstat, scancel, sprio, sstat

Per ottenere informazioni sui job schedulati con *sbatch*, come ad esempio *job-ID*, *status*, *partition* e l'utilizzo del numero di slot, è possibile usare il seguente comando:

```
$ squeue -u username

      JOBID PARTITION     NAME     USER   ST       TIME     NODES  NODELIST(REASON)
        600    global     test   username  R   2-23:29:19       1   compute-0-1
        601    global     test   username  R      3:53:04       3   compute-0-[2-4]
        602    global     test   username  R         7:55       1   compute-0-8
```

Per ottenere ulteriori informazioni sullo stato di ogni specifico job usare:

```
$ scontrol show job 600
JobId=600 JobName=test
   UserId=test(500) GroupId=test(500) MCS_label=N/A
   Priority=30937 Nice=0 Account=test QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=04:07:05 TimeLimit=8-08:00:00 TimeMin=N/A
   SubmitTime=2018-02-13T10:25:11 EligibleTime=2018-02-13T10:25:11
   StartTime=2018-02-13T10:25:12 EndTime=2018-02-21T18:25:12 Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=global AllocNode:Sid=casperlogin:30786
   ReqNodeList=(null) ExcNodeList=compute-0-8
   NodeList=compute-0-[2-4]
   BatchHost=compute-0-2
   NumNodes=3 NumCPUs=96 NumTasks=96 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=96,mem=240G,node=3
   Socks/Node=* NtasksPerN:B:S:C=32:0:*:* CoreSpec=*
   MinCPUsNode=32 MinMemoryNode=40G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   Gres=(null) Reservation=(null)
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/home/test/test.bin
   WorkDir=/home/test/
   StdErr=/home/test/job_600.log
   StdIn=/dev/null
   StdOut=/home/test/job_600.log
   Power=
```

Per ottenere informazioni sull'intero cluster e su tutti i job in esecuzione usare:

```
$ squeue
```

Lo stato delle partizioni può essere verificato con:

```
$ sinfo

PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
global*        up 10-00:00:0      3    mix compute-1-[9,18,23]
global*        up 10-00:00:0     25  alloc compute-1-[1-8,10,12-17,19-22,24-29]
global*        up 10-00:00:0      1   idle compute-1-11
cuda           up 10-00:00:0      1  alloc compute-1-14
bioeda         up   infinite      1  alloc compute-1-15
desal          up   infinite      3  alloc compute-1-[16-17,26]
e3             up   infinite      1    mix compute-1-18
e3             up   infinite      4  alloc compute-1-[19-22]
srg            up   infinite      1    mix compute-1-23
srg            up   infinite      1  alloc compute-1-24
bluen          up   infinite      1  alloc compute-1-25
small          up   infinite      1  alloc compute-1-27
nqs            up   infinite      2  alloc compute-1-[28-29]
maintenance    up   infinite      3    mix compute-1-[9,18,23]
maintenance    up   infinite     25  alloc compute-1-[1-8,10,12-17,19-22,24-29]
maintenance    up   infinite      1   idle compute-1-11
```

o utilizzando:

```
$ sjstat

Scheduling pool data:
-------------------------------------------------------------
Pool        Memory  Cpus  Total Usable   Free  Other Traits
-------------------------------------------------------------
global*    128752Mb    32      4      4      0  local,public,amd6276
global*    128752Mb    32      3      3      0  local,private,amd6274
global*    128752Mb    32      1      1      1  local,private,amd6128
global*    128752Mb    32      3      3      3  local,private,amd6276
global*    128752Mb    32      5      5      5  local,private,amd6376
lisa       128752Mb    32      1      1      0  local,private,amd6274
bioeda     128752Mb    32      2      2      0  local,private,amd6274
bioeda     128752Mb    32      1      1      1  local,private,amd6128
nqs        128752Mb    32      3      3      3  local,private,amd6276
modena     128752Mb    32      2      2      2  local,private,amd6376
pt-erc     128752Mb    32      3      3      3  local,private,amd6376
maintenan  128752Mb    32      4      4      0  local,public,amd6276
maintenan  128752Mb    32      3      3      0  local,private,amd6274
maintenan  128752Mb    32      1      1      1  local,private,amd6128
maintenan  128752Mb    32      3      3      3  local,private,amd6276
maintenan  128752Mb    32      5      5      5  local,private,amd6376

Running job data:
---------------------------------------------------------------------------
JobID    User      Procs Pool      Status        Used  Master/Other
---------------------------------------------------------------------------
```

Per visualizzare la priorità di tutti i componenti di tutti dei job usare:

```
$ sprio

          JOBID   PRIORITY        AGE  FAIRSHARE    JOBSIZE  PARTITION
```

opzioni utili:

`--jobs={jobID}`	Visualizza la priorità del/i job specificati (separati da virgola).
`--users={username}`	Visualizza la priorità dei job di uno più utenti specificati (separati da virgola).

Per arrestare e rimuovere un job da una partizione usare:

```
$ scancel jobID
```

Per raccogliere statistiche sui job correntemente in esecuzione è possibile utilizzare:

```
$ sstat --format=JobID,MaxRSS,AveRSS,AveCPU,NTask -j $JOBID --allsteps
       JobID     MaxRSS     AveRSS     AveCPU   NTasks
------------ ---------- ---------- ---------- --------
JOBID.0           3248K      3248K  00:00.000        1
```

Il comando precedente funziona solo per i job eseguiti attraverso la modalità interattiva `srun`, per i job tipo batch aggiungere `.batch` al `$jobID` specificato, come nel seguente esempio:

```
$ sstat --format=JobID,MaxRSS,AveRSS,AveCPU,NTask -j ${JOBID}.batch --allsteps
       JobID     MaxRSS     AveRSS     AveCPU   NTasks
------------ ---------- ---------- ---------- --------
${JOBID}.batch       9488K      9488K  07:24.000        1
```

Sessioni interattive sui nodi di computazione

Ogni volta si renda necessario l'uso di una sessione interattiva su un nodo di calcolo è utile l'uso del comando *srun*. Questo è il solo e unico metodo consentito per ottenere una sessione remota su uno dei nodi di computazione.

Dal nodo di login eseguire il comando con la seguente sintassi;

```
$ srun --nodes=1 --tasks-per-node=1 --pty /bin/bash
```

Le opzioni del comando corrispondono alle direttive presenti in uno script sbatch (vedere *sbatch*).

L'uso di *srun* viene rigorosamente monitorato come definito dalla policy SRUN ABUSE SUPPRESSION (vedere regolamento generale).

Un abuso è individuato quando una sessione *srun* viene contrassegnata come in esecuzione 'R' senza che però stia effettivamente eseguendo alcun calcolo. Questo potrebbe rendere incerta la policy di schedulazione.

Il meccanismo con cui la sessione *srun* "malevola" viene soppressa funziona nel seguente modo: tutti i giorni alle 3:00am per ogni *srun* in esecuzione viene calcolato il rapporto tra la media di utilizzo della CPU e il numero di CPU richieste al tempo di sottomissione, se questo valore è inferiore a una certa soglia il job viene terminato tramite *scancel*.

Sessioni grafiche e X11 Forwarding

Tramite il comando *srun* è possibile aprire una sessione grafica interattiva su un nodo computazionale. Per abilitare questa funzionalità occorre inizializzare *X11 Forwarding* alla prima connessione verso il nodo di login usando il comando:

```
$ ssh -YC user_name@{login-node}
```

successivamente usare il comando sotto specificato per ottenere una nuova sessione su uno dei nodi computazionali

```
[Casper] $ srun --nodes=1 --tasks-per-node=1 --x11 --pty /bin/bash
```

e infine invocare il programma che si intende utilizzare.

Il manuale di riferimento completo è disponibile qui: https://slurm.schedmd.com/

Miniconda Virtual Environment per Python

Miniconda è un package manager e environment manager ideato per permette di installare, eseguire e aggiornare packages e le loro dipendenze. Il tool mette a disposizione un virtual environment che permette l'esecuzione di diverse versioni di Python installate localmente nel profilo dell'utente.

- **Step 1: Scaricare e installare Miniconda**

  Effettuare l'accesso e verificare di essere all'interno della vostra cartella home usando il comando

  ```
  pwd
  ```

  Se l'output è del tipo /home/[nomeutente] potrete procedere con gli step successivi, altrimenti digitate

  ```
  cd /home/[nomeutente]
  ```

  Scaricare l'installer per miniconda (specificando la versione desiderata) usando il seguente comando, verrà creato il file *Miniconda3-py39_4.10.3-Linux-x86_64.sh* nella propria home.

  ```
  wget https://repo.anaconda.com/miniconda/Miniconda3-py39_4.10.3-Linux-x86_64.sh
  ```

  Al termine del comando verificate che sia stato scaricato il suddetto file .sh

  ```
  ls -l
  ```

  Applicare i permesi di esecuzione ed eseguite il file .sh

  ```
  chmod 755 ./[nomefile].sh
  ./[nomefile].sh
  ```

  A questo punto il programma di installazione vi chiederà di premere INVIO e vi mostrerà una licenza d'uso.
  Premere INVIO fino in fondo per scorrere l'intera licenza d'uso fino a quando il programma vi chiederà una conferma. Inserite 'yes' e premete INVIO.

  Adesso vi chiederà di confermare il percorso di installazione, se non avete particolari esigenze è sufficiente premere INVIO e confermare quella di default.
  Il processo di installazione a questo punto avrà inizio e potrebbe richiedere anche alcuni minuti.

  Al termine dell'installazione verrà visualizzato un messaggio dove l'installer chiede se si desidera inizializzare Miniconda, digitare 'yes' e premere INVIO.

- **Step 2: Configurare Miniconda e verificare la corretta installazione**

  Adesso è necessario modificare il file .bash_profile usando un editor di testo (In questo esempio useremo l'editor nano)

  ```
  nano ~/.bash_profile
  ```

  Nota: se il carattere non è presente sulla vostra tastiera potete usare la shortcut `CTRL + 126` (Win) oppure `AltGr + ì`(Linux).
  Una volta aperto il file dovrete aggiungere in coda alle variabili già presenti la seguente riga

  ```
  export "PATH=~/miniconda3/bin:$PATH"
  ```

  Per salvare le modifiche effettuate premere `CTRL + O` e premere INVIO, dopo per uscire premere `CTRL + X` Verifichiamo adesso che l'installazione sia andata a buon fine digitando:

  ```
  conda --v
  ```

  Se l'installazione è andata a buon fine visualizzerete correttamente il numero della versione di Miniconda installato (es. conda 4.10.3)

- **Installare una versione alternativa di Python**

  Visualizzare le versioni di Python disponibili per l'installazione digitando:

  ```
  conda search python
  ```

  Una volta appuntata la versione che ci interessa sarà necessario creare un nuovo ambiente virtuale usando la seguente sintassi:

  ```
  conda create -n [NomeAmbiente] python=X.X.X [packages]
  ```

  ad esempio:

  ```
  conda create -n py39 python=3.9 anaconda
  ```

  Verrà creato un nuovo ambiente virtuale chiamato py39, dove *3.9* è la versione che si intende usare e *anaconda* sono i moduli che si intendono installare. Il modulo *anaconda* comprende molti moduli comunemente usati con Python, per una lista dettagliata visitare `https://docs.anaconda.com/anaconda/packages/pkg-docs/`.
  Si noti che è possibile installare uno o più moduli contemporaneamente in fase di creazione dell'ambiente, ad esempio è possibile digitare

  ```
  conda create -n py39 python=3.9 numpy tensorflow
  ```

  In seguito verrà illustrato come aggiungere ulteriori packages ad ambienti virtuali esistenti. Una volta avviata la procedura, lo script ci chiederà una conferma di installazione quindi scriviamo y e premiamo INVIO, non ci resta che attendere la fine del processo di installazione che potrebbe richiedere anche alcuni minuti. Per permettere a conda di eseguire i comandi activate o deactivate digitiamo:

  ```
  source ~/.bashrc
  ```

  Ultimato il processo dovremo attivare l'ambiente digitando:

  ```
  conda activate -n [nomeAmbiente]
  ```

  che attiverà il nostro ambiente virtuale.

- **Installare packages in un ambiente virtuale esistente**

  Per installare i packages è necessario innanzitutto individuare il virtual environment su cui è necessario apportare modifiche. Per visualizzare la lista di tutti gli ambienti virtuali creati digitare:

  ```
  conda info -e
  ```

  Adesso è sufficiente digitare il seguente comando, sostituendo a [nomeAmbiente] il nome dell'ambiente virtuale e a [packages] la lista di tutti i moduli che si desidera installare separati da spazio:

  ```
  conda install -n [nomeAmbiente] [package1] ... [packageN]
  ```

  Se si volesse specificare una versione specifica di un package è sufficiente digitare il nome del modulo seguito da `=X.X` dove `X.X` è il numero della versione (ad esempio *numpy=1.9.3*).
  Per cercare un pacchetto da installare digitare:

  ```
  conda search [nomePacchetto]
  ```

- **Comandi utili**

Comando	Descrizione
conda create -n [nomeAmbiente] [packages]	Crea un nuovo ambiente virtuale e installa i packages specificati
conda search [package]	Cerca il package sulle repository di Miniconda
source /.bashrc	Pemette alla shell di eseguire i comandi di Miniconda dalla cartella di installazione di Miniconda
conda activate [nomeAmbiente]	Attiva l'ambiente virtuale specificato
conda deactivate	Disattiva l'ambiente virtuale corrente
conda list -n [nomeAmbiente]	Elenca i pacchetti installati nell'ambiente virtuale specificato

Per altre informazioni su comandi o su Miniconda visitare il sito
`https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html`

Eseguire script Matlab su più nodi

Per eseguire calcoli in parallelo, Matlab mette a disposizione delle apposite funzioni (parfor, parpool, etc..). Di default, questi comandi vengono eseguiti in parallelo sullo stesso nodo. Per poter eseguire il calcolo su più nodi (utilizzando dunque un maggior numero di core, che in matlab vengono definiti Workers) è necessario procedere con i passaggi seguenti. Qui si fa riferimento a Matlab versione 2020b.

- **Step 1: Connessione al cluster e avvio di Matlab**

  Per avviare l'interfaccia grafica di Matlab è necessario innanzitutto collegarsi al cluster con il seguente comando:

  ```
  ssh -YC username@hactarlogin.polito.it
  ```

  e procedere con l'inserimento delle credenziali. Successivamente, verificare la presenza del modulo matlab/2020b (necessario per eseguire in parallelo il job):

  ```
  module av | grep matlab
  ```

  Se il modulo compare tra i disponibili, caricarlo eseguendo il comando

  ```
  module load matlab/2020b
  ```

  Per far partire l'interfaccia grafica direttamente sulla nostra macchina, lanciare il seguente comando

  ```
  matlab
  ```

  Questo ci consentirà di navigare facilmente nelle impostazioni del programma.

- **Step 2: creazione del profilo Cluster**

  L'interfaccia grafica di Matlab potrebbe essere molto più lenta del solito, questo perché il programma non viene eseguito direttamente sulla nostra macchina, ma si tratta di una interazione con il cluster remoto. Recarsi nel seguente menù

  ```
  Home -> Environment -> Parallel -> Create and Manage Clusters
  ```

  Tramite questa pagina è possibile creare (o modificare) un profilo personalizzato per l'esecuzione dei vostri script Matlab. Per iniziare selezionamo "Add cluster Profile" e selezioniamo Slurm fra le opzioni. Dopo la creazione del profilo possiamo rinominarlo (se vogliamo) e poi procedere con l'aggiunta delle opzioni necessarie ad abilitare il parallelismo degli script su Hactar. Click su "Edit" e modificare le seguenti opzioni:

  ```
  JobStorageLocation: una destinazione a scelta in /home/username
  NumWorkers: un numero fra 8 e 256 (1 worker = 1 core)
  ClusterMatlabRoot: /scratch/MATLAB/2020b
  ```

  Una volta salvate le nuove impostazioni è necessario validare la configurazione. Spostarsi sulla scheda "Validation", impostare eventualmente il numero di worker da usare per questa fase di validazione (valore opzionale) e cliccare sul bottone "Validate". Se il processo è andato a buon fine il profilo creato è corretto; è possibile chiudere la finestra. Si consiglia, a questo punto, di impostare il profilo creato come predefinito andando su:

  ```
  Home -> Environment -> Parallel -> Select a default cluster
  ```

  e selezionando il profilo precedentemente creato.

- **Step 3: Esecuzione script**

  Giunti a questo step, è possibile mandare in esecuzione i propri script (che dovranno usare le opportune istruzioni "par*") in paralello utilizzando il numero di core specificato nel profilo.

Distribuzione dei job tramite MPI

SLURM mette a disposizione una gran varietà di opzioni per configurare come le risorse dovrebbero venire riservate al job. La seguente tabella riassume le principali:

`--nodes`	Ogni server corrisponde a un nodo e il loro numero viene specificato tramite la direttiva `--nodes`
`--ntasks`	I task corrispondono al numero di processi MPI e il loro numero vengono specificato tramite questa direttiva
`--ntasks-per-node`	Offre la possibilità di controllare il numero di task per singolo nodo
`--cpus-per-task`	Imposta quante CPU per task saranno riservate, in unione con OpenMP rappresenta il numero di core per singolo processo OpenMP

Esempi di script sbatch MPI

MPI case 1 [mpi1.sbatch]

```
#!/bin/bash
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
```

Questo esempio richiede 16 task, corrispondenti a 16 processi MPI, a cui viene associata una CPU per task. Tutti i 16 task sono vincolati a essere eseguiti su un sigolo nodo di computazaione.

MPI esempio 2 [mpi2.sbatch]

```
#!/bin/bash
#SBATCH --ntasks=32
#SBATCH --cpus-per-task=1
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=16
```

In questo esempio sono richiesti 32 task suddivisi tra 2 nodi, con 16 task per ogni nodo.

MPI esempio 3 [mpi3.sbatch]

```
#!/bin/bash
#SBATCH --ntasks=32
```

In questo terzo esempio vengono richiesti 32 task e la scelta della loro distribuzione all'interno del cluster è lasciata allo scheduler.

Casi d'uso MPI e OpenMP [mpiOpenMP.sbatch]

```
#!/bin/bash
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=4
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
```

In questo caso viene eseguita un'applicazione che, per ogni processo (o rank) MPI, usa core multipli. Vengono richiesti 16 task (`--ntasks=16`) e 4 core per task (`--cpus-per-task=4`), quindi 16*4=64 core totali. I 16 task vengono ripartiti tra 4 nodi (`--nodes=4`) con 4 task per nodo (`--ntasks-per-node=4`).

Esempio CUDA

Esempio GPU CUDA [cuda.sbatch]

```
#!/bin/bash
#SBATCH --time=24:00:00
#SBATCH --ntasks=1
#SBATCH --partition=cuda
#SBATCH --gres=gpu:1
#SBATCH --job-name=cudaExample
#SBATCH --mem=10GB
#SBATCH --mail-type=ALL
##
# load cuda module
module load nvidia/cudasdk/9.2
```

Questo esempio richiede 1 GPU, per utilizzare le funzionalità CUDA è necessario caricare il modulo CUDA.

Scheduling e priorità dei job

In tutti i cluster del centro HPC@POLITO le risorse vengono gestite dallo scheduler, questo si occupa di assegnare i job alle risorse disponibili sulla base del carico corrente e delle caratteristiche stesse del job. L'algoritmo utilizzato per gestire i job in coda è di tipo multifactor con backfill. Tale meccanismo entra in azione in tutti quei casi in cui le risorse richieste per l'avvio dei job non sono immediatamente disponibili e quindi i job permangono in coda, ordinati in ordine di priorità. I job in testa alla coda partiranno non appena le risorse richieste saranno disponibili, grazie all'algoritmo di backfill lo scheduler avvia job a priorità minore se ciò non causa ritardi nell'avvio dei job a priorità maggiore. Affinché questo algoritmo funzioni nel modo più efficiente possibile è richiesto che ai job venga assegnato un timelimit il più possibile aderente all'effettivo tempo di runtime.

Calcolo della priorità

I fattori che concorrono al calcolo della priorità sono i seguenti:

- **Age Factor:** valore direttamente proporzionale al tempo di permanenza in coda.
- **Job Size Factor:** valore direttamente proporzionale alla grandezza(CPU, MEMORY, TIME) del job, in questo modo è garantito che i job che richiedono molte risorse riescano a partire.
- **Partition Factor:** valore utilizzato per il funzionamento delle partizioni prioritarie assieme alla pre-emption delle risorse.
- **Fair-share Factor:** valore calcolato rispetto alla quantità di risorse consumate in rapporto alla totalità delle risorse disponibili su una finestra mobile di 30 giorni.
- **TRES Factor:** valore assegnato qualora i job richiedano determinate risorse (GPU).
- **QOS Factor:** valore calcolato sull'efficienza nell'utilizzo delle risorse da parte del singolo utente su una finestra mobile di 30 giorni.

QOS Factor

Con il fine di incentivare l'uso efficiente e consapevole delle risorse a disposizione è stato introdotto tra le metriche il QOS Factor.
L'algoritmo utilizzato per determinare il QOS Factor è il seguente: per ogni utente viene valutata l'efficienza nell'uso delle risorse (CPU, memoria, tempo d'esecuzione) di ogni job sottomesso su una finestra mobile di 30 giorni eseguendo il rapporto di quanto effettivamente consumato con il richiesto in fase di sottomissione. I valori calcolati vengono poi normalizzati rispetto al valore massimo ottenuto dagli utenti per il periodo in esame.
L'utilizzo di tale metrica è volto a premiare gli utenti che utilizzano le risorse in maniera più efficiente andando a ridurre la frequenza delle problematiche di seguito elencate:

- **job con timelimit non allineato al tempo d'esecuzione:** inefficienza dell'algoritmo backfill, non è possibile sfruttare porzioni di risorse non utilizzate dal job a maggior priorità nell'attesa che la totalità delle risorse necessarie alla sua esecuzione si liberino con una conseguente diminuzione del throughput del sistema.
- **job con richieste di memoria non allineata a quanto effettivamente consumato:** sottoutilizzo delle risorse e maggior tempo d'attesa in coda, sia per quanto riguarda il job stesso, sia per quanto riguarda gli altri job in coda. Non ultimo danneggia pesantemente il sistema di pre-emption attivo sulle code prioritarie in quanto la memoria è una risorsa su cui la policy di sospensione non ha alcun impatto (la memoria allocata da processi sospesi continua ad essere allocata anche in caso di sospensione).
- **job con richieste di CPU non allineata a quanto effettivamente consumato:** sottoutilizzo delle risorse e maggior tempo d'attesa in coda, sia per quanto riguarda il job stesso, sia per quanto riguarda gli altri job in coda.

Per migliorare l'efficenza nell'utilizzo delle risorse è possibile usare il comando **sacct** e il comando **seff** per l'interrogazione del sistema di accounting, o in alternativa utilizzare le statistiche riportate nelle email di notifica al termine dei job.

Allocazione memoria

Slurm tratta la memoria come una risorsa consumabile, per questa ragione è necessario essere il più precisi possibile nel dimensionarne la richiesta. Richieste non coerenti con l'effettivo uso penalizzano fortemente le prestazioni generali del sistema, sia in termini di utilizzo delle risorse, sia di tempi medi di attesa in coda. Se la necessità di memoria non è particolarmente elevata si può utilizzare il valore di default del sistema (1 GB per core) semplicemente omettendo la direttiva in cui si specificano le richieste relative alla memoria (`--mem` o `--mem-per-cpu`). Per usi più intensi la seguente sezione illustrerà come dimensionare correttamente la richiesta di memoria.
Se risulta difficoltoso o impossibile stimare accuratamente per via analitica la quantità di memoria, la migliore soluzione è ricavare le informazioni deducendole da job simili completati in precedenza.

seff

Per fare questo è possibile utilizzare il tool **seff** dal nodo di login:

```
[~]xxxxxxxx@hactarlogin$ seff $JOBID
Job ID: $JOBID
Cluster: hactar
User/Group: xxxxxxxx/xxxxxxxxx
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 10
CPU Utilized: 10:17:10
CPU Efficiency: 10.00% of 4-06:53:00 core-walltime
```

```
Job Wall-clock time: 10:17:18
Memory Utilized: 8.43 GB
Memory Efficiency: 86.30% of 9.77 GB
```

Questa utility viene anche usata per generare un report che, al termine del job, verrà incluso nella e-mail di notifica nel caso in cui sia stata fatta richiesta durante la sottomissione. Nell'esempio prcedente l'efficienza nell'uso di memoria è dell'86.30%, un valore del tutto ragionevole.

sacct

Un'ulteriore possibilità è interrogare il sistema di accounting tramite **sacct**:

```
[~]xxx@hactarlogin$ sacct --format \
JobID,JobName,Partition,AveRSS,MaxRSS,ReqMem,State,AllocCPUS -j $JOBID

       JobID    JobName  Partition     AveRSS     MaxRSS     ReqMem      State  AllocCPUS
------------ ---------- ---------- ---------- ---------- ---------- ---------- ----------
$JOBID            $JOBNAME  global                          1000Mc  COMPLETED         10
$JOBID.batch      batch              8596096K   8836936K     1000Mc  COMPLETED         10
```

dove:

`ReqMem`	E' la memoria richiesta per SLURM. Se il valore ha suffisso *Mc* indica la memoria per core in MB, se ha suffisso *Mn* indica la memoria per nodo in MB.
`MaxRSS`	E' il massimo di memoria usata. Il valore è applicato direttamente per i job in esecuzione su singolo nodo, mentre per job in esecuzione su nodi multipli indica il valore massimo di memoria usata su singolo nodo in cui il job sta girando, il valore deve essere normalizzato rispetto al numero di core usati sul nodo.
`AveRSS`	Indica l'utilizzo medio di memoria
`Elapsed`	Indica il tempo di esecuzione.

Nota: solo le statistiche su job già terminati sono da considerarsi affidabili.

Per ottenere la quantità di memoria richiesta per core è possibile: dividere il valore *MaxRss* per il numero di core richiesti per nodo o usati sul nodo se la distribuzione dei processi non è uniforme tra i nodi, oppure specificare direttamente il valore di *MaxRSS* se si specifica l'opzione `--mem`.
Ad esempio, per fare una richiesta con un margine del 20% in più rispetto al valore di memoria rilevato:

`--mem-per-cpu=` *(MaxRSS / cores-per-node) * 1.2*
`--mem=` *MaxRSS * 1.2*

Si raccomanda a tutti gli utenti di verificare e validare le proprie richieste di memoria in tutti i casi in cui la quantità specificata nella sottomissione sia superiore al valore di default (1GB per core). Verranno eseguiti controlli a campione sui job attivi, tutti i job in cui la richiesta di memoria sarà il doppio di quella effettivamente in uso verranno forzosamente terminati senza alcun preavviso.

Nel caso in cui il proprio job fallisca, o venga terminato a causa di limiti di memoria, nel file di log relativo al job terminato saranno presenti alcune linee simili alle seguenti:

```
slurmstepd: error: Job $JOBID exceeded memory limit (2773388 > 1024000), being killed
slurmstepd: error: *** JOB $JOBID ON compute-x-x CANCELLED AT xxxx-xx-xxTxx:xx:xx ***
```

In questo caso ad esempio si dovrebbe elevare il limite richiesto da 1GB a 3 GB (il minimo valore sarebbe di 2773388 KB, ma è consigliato un certo margine).

Sistema di Module Environment

Il software *Modules* (*TACC Lmod*) consente la modifica dinamica delle variabili di ambiente durante una sessione utente.
L'uso di questo software è fortemente consigliato negli script *sbatch* e nelle sessioni *srun* dato che fornisce un modo semplice per utilizzare versioni differenti della stessa applicazione; in pratica consente agli utenti di esportare le variabili di ambiente necessarie al corretto funzionamento.
Per il caricamento dei moduli necessari aggiungere i comandi modules direttamente negli script sbatch.

Opzioni di uso comune per *module*:

`module list`	Elenca i moduli caricati
`module avail`	Elenca i moduli disponibili
`module load {nome-modulo}`	Carica un modulo
`module unload {nome-modulo}`	Rimuove un modulo
`module purge`	Rimuove tutti i moduli

Esempio:

```
$ module avail

-------------------- /opt/ohpc/pub/moduledeps/gnu-mvapich2 ---------------------
   adios/1.11.0    mpiP/3.4.1              petsc/3.7.5        scorep/3.0
   boost/1.63.0    mumps/5.0.2             phdf5/1.8.17       sionlib/1.7.0
   fftw/3.3.4      netcdf-cxx/4.3.0        scalapack/2.0.2    superlu_dist/4.2
   hypre/2.11.1    netcdf-fortran/4.4.4    scalasca/2.3.1     tau/2.26.1
   imb/4.1         netcdf/4.4.1.1          scipy/0.19.0       trilinos/12.10.1

------------------------- /opt/ohpc/pub/moduledeps/gnu -------------------------
   gsl/2.2.1      mpich/3.2           ocr/1.0.1          pdtoolkit/3.23
   hdf5/1.8.17    mvapich2/2.2 (L)    openblas/0.2.19    superlu/5.2.1
   metis/5.1.0    numpy/1.11.1        openmpi/1.10.6

------------------------- /opt/ohpc/pub/modulefiles --------------------------
   autotools        (L)    gnu/5.4.0  (L)    pmix/1.2.3
   clustershell/1.8        ohpc       (L)    prun/1.2        (L)
   cmake/3.9.2             papi/5.5.1        singularity/2.4

----------------------- /share/apps/casper-modulefiles -----------------------
   blender/2.79                       intel/python/2.7/2017.3.053
   converge/2.3.22                    intel/python/3.5/2017.3.052
   fire/2014.2                        lammps/16Feb16
   intel/libraries/daal/2017.4.239    matlab/2017b
   intel/libraries/ipp/2017.3.196     quantum-espresso/5.4.0
   intel/libraries/mkl/2017.4.239     quantum-espresso/6.2.1      (D)
   intel/libraries/mpi/2017.4.239     starccm+/12.06.011
   intel/libraries/tbb/2017.8.239

  Where:
   L:  Module is loaded
   D:  Default Module
```

Seguono una serie di esempi sull'utilizzo dei moduli.

Caricare i moduli necessari per eseguire *Matlab*:

```
$ module load matlab/2017b
```

Caricare automaticamente il modulo corretto a seconda del cluster in cui il job viene sottomesso.
La versione di *starccm* è differente sui due cluster:
su CASPER è installata la versione "starccm+/12.02.010";
su HACTAR è installata la versione "starccm+/12.02.011".

Inserire nel file di sottomissione le seguenti righe di codice in modo da caricare il modulo per la corretta versione del software:

```
if [ $SLURM_CLUSTER_NAME == "hactar" ]
then
    module load starccm+/12.02.011
else
    module load starccm+/12.02.010
fi
```

Sistema di monitoraggio Ganglia

Ganglia è un sistema di monitoraggio, scalabile e distribuito, per sistemi di calcolo ad alte prestazioni, cluster e reti. Il software è usato per visualizzare sia l'andamento in tempo reale che statistiche registrate di diversi parametri, quali carico medio di CPU o utilizzo della rete per i vari nodi.

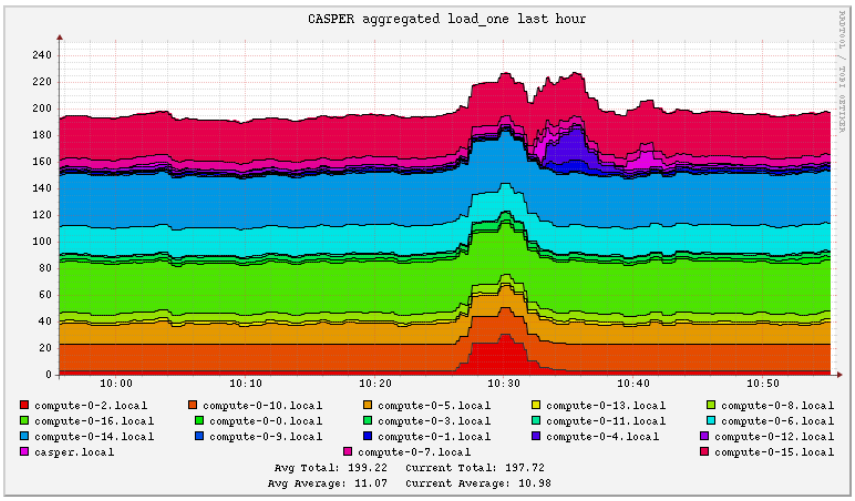

Figura 3: Ganglia - Ganglia example

Ganglia mette a disposizione un'interfaccia web che fornisce informazioni riguardanti le attività del cluster. E' raggiungibile ai seguenti link:

[Casper] *http://casper.polito.it/ganglia/*

[Hactar] *http://hactar.polito.it/ganglia/*

[Legion] *http://legion.polito.it/ganglia/*

APPENDICE A: esempi di script sbatch

Segue una lista di script di esempio per alcune delle applicazioni installate sui cluster

Matlab [matlab.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=hh:mm:ss
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=1024M

module load matlab/2017b

matlab -r test_matlab
```

Blender [blender.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=hh:mm:ss
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=1024M

module load blender/2.79

blender -noaudio -b BMW1M.blend -o BMW1M_out -F PNG -f 1 -t${SLURM_NTASKS}
```

Converge [converge.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=24:00:00
#SBATCH --ntasks=96
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=2048M

module purge
module load converge/2.3.22

echo Starting on $SLURM_JOB_NODELIST with $SLURM_NTASKS processors

mpirun --bind-to core converge-2.3.22-openmpi-linux-64 $inFile > output
```

Quantum-espresso [qe.sbatch]

```
#!/bin/bash
#SBATCH --job-name=job_name
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=24:00:00
#SBATCH --nodes=3
#SBATCH --ntasks-per-node=32
#SBATCH --exclude=compute-0-8
#SBATCH --output=job_name_%j.log
#SBATCH --mem-per-cpu=2048M

module purge

module load quantum-espresso/6.2.1

CASE_IN="ausurf.in"

echo Starting on $SLURM_JOB_NODELIST with $SLURM_NTASKS processors

prun pw.x -ntg 2 -ndiag 16 -input $CASE_IN
```

STAR-CCM+ [starccm.sbatch]

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=24:00:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=32
#SBATCH --output=%x_%j.log
#SBATCH --error=%x_%j.err
#SBATCH --mem-per-cpu=1G

module purge
module load starccm+/12.06.011

#----------------------------
LICENSE_FILE='1999@flex.cd-adapco.com'
LIC_KEY='chiave_pod'
#----------------------------

echo '' > machinefile
for node in $(scontrol show hostnames $SLURM_JOB_NODELIST)
        do
        echo ${node}:${SLURM_NTASKS_PER_NODE} >> machinefile
        done

starccm+ -power -licpath $LICENSE_FILE -podkey $LIC_KEY -np $SLURM_NTASKS \
         -machinefile machinefile -mpi intel -mpiflags "-bootstrap slurm" \
         -fabric ibv -fabricverbose -batch-report -batch run $SLURM_JOB_NAME.sim
```

LAMMPS [lammps.sbatch]

```
#!/bin/bash
#SBATCH --job-name=nome_job
#SBATCH --mail-type=ALL
#SBATCH --mail-user=nome.cognome@polito.it
#SBATCH --partition=global
#SBATCH --time=20:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --constraint=amd6274|amd6276|amd6376
#SBATCH --output=nome_job_%j.log
#SBATCH --mem-per-cpu=2048

module purge
module load prun/1.2
module load lammps/16Feb16

CASE_IN="in.chain"
echo Starting on $SLURM_JOB_NODELIST with $SLURM_NTASKS processors

prun lmp_mpi -echo screen -in $CASE_IN
```

APPENDICE B: generare script di sottomissione da file *csv*

Per task avanzati che richiedono una esecuzione multipla dello stesso programma, ma con parametri differenti, si suggerisce l'uso del seguente script in grado di generare file sbatch multipli a partire da un file di testo contenente tutti i parametri.

[generate.sh]

```
#!/bin/bash
#
list=$(cat ./parameters.csv)
#
for i in $list
    do
    parameter1=$(echo $i|cut -f 1 -d ',')
    parameter2=$(echo $i|cut -f 2 -d ',')
    cat << EOF > ./test-$parameter1-$parameter2.sbatch
    #!/bin/bash
    #SBATCH --job-name=job_name
    #SBATCH --mail-type=ALL
    #SBATCH --mail-user=nome.cognome@polito.it
    #SBATCH --partition=global
    #SBATCH --time=24:00:00
    #SBATCH --nodes=3
    #SBATCH --ntasks-per-node=32
    #SBATCH --output=job_name_%j.log
    #SBATCH --mem-per-cpu=2048M
    example.bin  $parameter1 $parameter2
    EOF
done
```

[parameters.csv]

```
A,1
A,2
B,1
B,2
```

APPENDICE C: container Singularity

Per offrire agli utilizzatori la massima flessibilità, è stata introdotta la possibilità di eseguire container di tipo *Singularity* all'interno del cluster.
Il manuale di riferimento completo è disponibile qui: http://singularity.lbl.gov/user-guide
Segue un esempio per creare ed eseguire un container con TensorFlow.

Step1: creare un Singularity recipe file

Ci sono svariate possibiltà per creare un container, una di queste è partire da una immagine Docker.
In questo esempio si inizia da un'immagine Docker NVIDIA.
Creare un file *tensorflowCentos.recipe* sul proprio PC con il seguente contenuto:

```
BootStrap: docker
From: nvidia/cuda:9.0-cudnn7-devel-centos7   # This is a comment

%runscript
   echo "Hello from tensorflow container"
   whoami

%post
   echo "Hello from inside the container"
   yum -y update && yum install -y epel-release
   yum install -y python-pip python-devel
   pip2 install matplotlib h5py pillow tensorflow-gpu keras scikit-learn
```

Step2: build del container Singularity

Esistono diversi tipi di container, in questo caso si creerà un'immagine immutabile.

```
$ sudo singularity build  tensorflowCentos.img tensorflowCentos.recipe
```

Step3: copia del container sul cluster

```
$ scp tensorflowCentos.img {username}@{login-node}:/home/{login-node}/tensorflowCentos.img
```

Step4: creare lo script di sottomissione

Accedere al nodo di login e creare il seguente file python nella propria home.
/home/{user_name}/helloTensorflow.py

```
import tensorflow as tf
hello = tf.constant("Hello, TensorFlow!")
sess = tf.Session()
print sess.run(hello)
exit()
```

Creare un file *singularity.sbatch* in base al seguente esempio:

```
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --ntasks=1
#SBATCH --partition=cuda
#SBATCH --gres=gpu:1
#SBATCH --job-name=singularityTest
#SBATCH --mail-type=ALL

module load singularity/2.4.5

singularity exec --nv tensorflowCentos.img bash -c 'python helloTensorflow.py'
```

Ora è possibile sottomettere il job con il comnado sbatch:

```
sbatch singularity.sbatch
```

APPENDICE D: Lavorare con Python

All'interno dei cluster del centro è possibile lavorare con Python. Sono disponibili le seguenti versioni, due di sistema facenti parti della distribuzione CentOS, e due versioni Intel Distribution for Python. Versioni di sistema disponibili tu tutti i cluster:

- Python 2.7.5
- Python 3.4.10

Per utilizzarle è sufficente chiamare l'eseguibile rispettivamente:

```
python
python3.4
```

Per le versioni di Intel è necessario caricare il modulo corrispondente, per maggiori informazioni su come utilizzare i moduli fare riferimento al capitolo Sistema di Module Environment. Per garantire migliori prestazioni raccomandiamo l'utilizzo delle versioni Intel python.
Una volta definita quale versione di python si intende utilizzare, per avere la massima flessibilità nella scelta dei moduli python gli utenti possono installare tutti i pacchetti di cui necessitano in un Virtualenv.

Virtualenv

Virtualenv è uno strumento che serve a isolare e gestire ambienti di sviluppo separati.
Con virtualenv puoi creare diversi ambienti virtuali distinti con relative dipendenze e pacchetti PyPI di terze parti.
Tutto questo, compresa l'esatta versione Python che utilizzi, finisce sotto una directory di ambiente (il cui percorso è personalizzabile).

VIRTUALENV CHEATSHEET	
crea	virtualenv env
attiva	source env/bin/activate
disattiva	deactivate

Python workflow

In questa sezione vediamo tutti i passi necessari eseguire uno script python all'interno di uno dei cluster dell' iniziativa HPC@POLITO.
Gli esempi di seguito forniti fanno riferimento al cluster Legion. Nell'esempio vedremo come installare il modulo "TensorFlow" e come eseguire un semplice script python.
Per eseguire con successo il nostro codice python possiamo scomporre il workflow in due parti principali:

1. Preparazione virtual environment
2. Esecuzione job python

Preparazione virtual environment

I seguenti passi sono da eseguire ogni qual volta necessitiamo di un ambiente di lavoro differente.
Per la creazione di un virtualenv è sufficente collegarsi al nodo di login del cluster.

```
ssh yourusername@legionlogin.polito.it
```

Una volta ottenuta una shell sul nodo di login è necessario caricare il modulo intel di python della versione che si intende utilizzare. Per visualizzare i moduli disonibili è possibile utilizzare il comando:

```
module avail
```

Una volta individuato il nome del modulo, nel seguente esempio useremo il modulo "intel/python/3/2019.4.088", bisogna caricarlo con il comando:

```
module load intel/python/3/2019.4.088
```

Ora è possibile procedere con la creazione del virtualenv con il comando ("myvenv" è il nome utilizzato nel resto della guida, è possibille scegliere un qualsiasi nome per il proprio virtualenv):

```
virtualenv myvenv
```

Una volta teminata la creazione dobbiamo attivare il virtualenv, questa operazione sarà da eseguire sempre quando si intende utilizzare il virtualenv.

```
source myvenv/bin/activate
```

Se il virtualenv si attiva correttamente potete notare che apparirà il nome del virtualenv tra parentesi prima del prompt del terminale:

```
(myvenv) <yourusername>@legionlogin$
```

Ora è possibile procedere all'installazione dei moduli necessari per il nostro codice python. Per il nostro esempio installeremo il modulo "tensorflow-gpu", è sufficente utilizzare il comando pip:

```
pip install tensorflow-gpu
```

Per mostrare i pacchetti installati all'interno di un virtualenv è sufficente una volta attivato usare il comando:

```
pip list --local
```

Terminata l'installazione e verificato che i moduli necessari sono presenti si conclude la fase di preparazione del virtual environment.

Esecuzione job python

Per eseguire un job all'interno del cluster è necessario creare uno script di sottomissione (maggiori dettagli sbatch). Al suo interno è necessario inserire tutti i comandi necessari per l'esecuzione del job:

- direttive per lo scheduler
- load dei moduli
- attivazione virtualenv
- esecuzione codice

Potete creare il file di sottomissione ed editarlo con vim direttamente sul nodo di login, o se preferite potete crearlo sulla vostra postazione e copiarlo successivamente all'interno della vostra home sul cluster. Procedete alla creazione di un file "mysbatch.sbatch" con il seguente contenuto:

```
#!/bin/bash
#SBATCH --time=00:20:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --gres=gpu:1
#SBATCH --job-name=python-ex
#SBATCH --mail-type=ALL
#SBATCH --partition=cuda

###### 1 Load the module
module load nvidia/cudasdk/10.1
module load intel/python/3
```

```
# 2 Activate the virtual environment
source myvenv/bin/activate

# 3 Run the python script
python tf-mnist.py
```

Creiamo quindi il file python "tf-mnist.py" che vogliamo eseguire sulle risorse del cluster:

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Se abbiamo eseguito correttamente tutti i passaggi dobbiamo avere nella home del nostro account:

1. directory "myvenv"
2. file python "tf-mnist.py"
3. file di sottomissione "mysbatch.sbatch"

E' ora possibile sottomettere il job python con il comando:

```
sbatch mysbatch.sbatch
```