

SLURM - MANUALE UTENTE

- 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.davin@polito.it

Indice

1	Destinatari	2
2	Gentlemen's agreement	3
3	Regole per l'uso dei cluster	3
4	HPC@POLITO Academic Computing Center	4
4.1	Casper - Specifiche tecniche	4
4.2	Hactar - Specifiche tecniche	4
4.3	Storage - Specifiche tecniche	4
5	Primo accesso	5
6	Trasferimento file e storage	6
6.1	Home Storage	6
6.2	Storage ad alte prestazioni	7
6.3	Storage locale ai nodi	7
7	Uso del cluster	8
7.1	Controllo e sottomissione di un job	8
7.1.1	sbatch	8
7.1.2	squeue, sinfo, sjstat, scancel, sprio, sstat	9
7.2	Sessioni interattive sui nodi di computazione	13
7.3	Sessioni grafiche e X11 Forwarding	13
8	Distribuzione dei job tramite MPI	14
8.1	Esempi di script sbatch MPI	14
9	Esempio CUDA	15
10	Allocazione memoria	15
11	Sistema di Module Environment	17
12	Sistema di monitoraggio Ganglia	19
13	APPENDICE A: esempi di script sbatch	20
14	APPENDICE B: generare script di sottomissione da file <i>csv</i>	23
15	APPENDICE C: container Singularity	23

Destinatari

La corrente guida illustra l'utilizzo dei sistemi HPC@POLITO basati su scheduler SLURM. Il documento può essere utilizzato per entrambi 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*

Se si è familiari con lo scheduler SGE segue un elenco di cosa tenere a mente per l'utilizzo di SLURM:

1. SLURM non possiede i *parallel environment* (orte, shared o altri), per specificare come distribuire le risorse da utilizzare si fa uso delle direttive `--nodes` e `--tasks-per-node`.
2. Le code di SGE sono chiamate *partizioni* in SLURM.
3. SGE consente di specificare un limite massimo e un valore di memoria da pre-allocare per lo svolgimento del task; in SLURM i parametri della memoria specificano il suo reale utilizzo e influenzano la decisione su dove il job verrà avviato (`--mem` e `--mem-per-cpu`). Se non si è a conoscenza della quantità di memoria necessaria per il proprio programma, lasciare che lo scheduler utilizzi i propri valori di default.
4. Alcune variabili di ambiente sono cambiate (ad es. SGE `$JOB_ID`, in SLURM diventa `$SLURM_JOB_ID`, o SGE `$NSLOTS`, in SLURM diventa `$SLURM_NTASKS`), per informazioni complete fare riferimento alla documentazione ufficiale SLURM: <https://slurm.schedmd.com/sbatch.html>.

Per una tabella comparativa più dettagliata fare riferimento a <https://slurm.schedmd.com/rosetta.pdf>

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.), nonché 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. Sugeriamo 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
Potenza consumata	7 kW
Core computazionali	512
Numero di nodi	16
Memoria RAM totale	2.0 TB DDR3 REGISTERED ECC
OS	Centos 7.4 - OpenHPC 1.3.4
Scheduler	SLURM 17.11

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
Numero di nodi	29
Memoria RAM totale	3.7 TB DDR4 REGISTERED ECC
OS	CentOS 7.4 - OpenHPC 1.3.4
Scheduler	SLURM 17.11

Storage - Specifiche tecniche

Home Storage	140 TB on RAID 6, throughput prossimo a 200 MB/s
Lustre Storage	87 TB. throughput maggiore di 2.2 GB/s
Rete di interconnessione	Ethernet 10 Gb/s

Primo accesso

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

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:

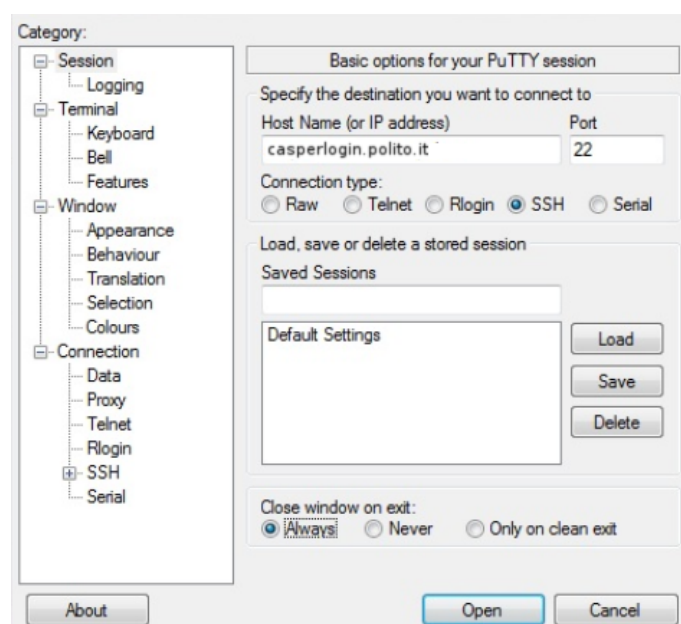


Figura 1: Putty - Configurazione per l'accesso da sistemi Windows

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

```
$ passwd
```

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.

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

Home Storage

Una volta effettuato l'accesso ci si troverà direttamente nella propria home directory attraverso l'*Home Storage*, accessibile su entrambe 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 1TB 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.

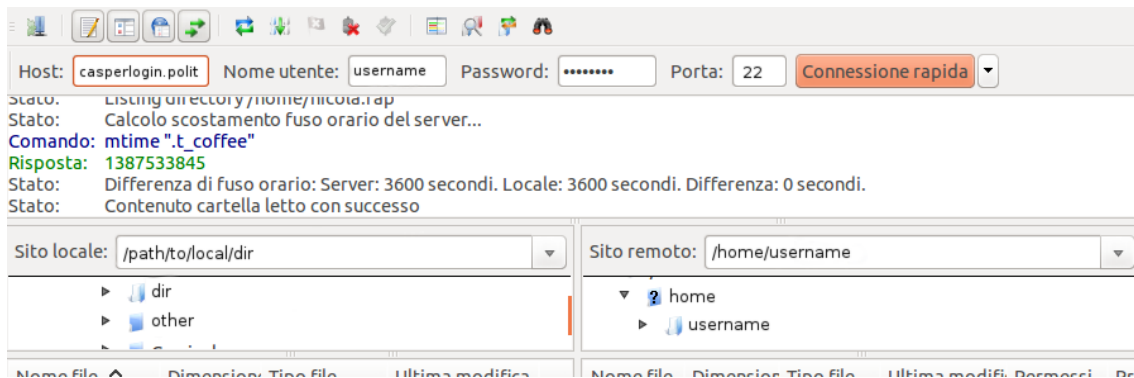


Figura 2: Configurazione FileZilla per il trasferimento file su {login-node}

Storage ad alte prestazioni

Tutti gli utenti possono accedere a un addizionale storage ad alte prestazioni. Questo storage è provvisto di file system Lustre in grado di incrementare le prestazioni di tutti i processi che fanno un uso intensivo di I/O sui file.

Questo storage rappresenta una *home directory* secondaria ed è accessibile al seguente percorso:

```
/work/username/
```

Per poter trasferire i file dal normale storage in uso allo storage ad alte prestazioni, sarà quindi sufficiente l'uso di *cp*:

```
$ cp /home/username/filename /work/username/filename
```

```
$ cp -r /home/username/directoryname /work/username/directoryname
```

Su questa unità di storage, per ogni utente, è riservato 1TB ulteriore di spazio disco (*soft quota*), ma in caso di necessità sarà possibile oltrepassare tale soglia fino a 1.5TB (*hard quota*) per non più di 7 giorni.

Per conoscere lo stato della propria quota usare il comando:

```
$ get-my-lustre-quota -h
```

Lo storage ha una capacità di 87TB e può raggiungere performance di 1.1GB/s se usato da CASPER e oltre 2.2 GB/s da HACTAR

Gli utenti possono richiedere un aumento della quota loro riservata, inviando una mail a hpc.davin@polito.it e includendo le opportune 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

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 (nodo *compute-1-14*), 2 slot GPU; la massima durata di un job è 10 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 cluster 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:

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

<code>--jobs={jobID}</code>	Visualizza la priorità del/i job specificati (separati da virgola).
<code>--users={username}</code>	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/>

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:

<code>--nodes</code>	Ogni server corrisponde a un nodo e il loro numero viene specificato tramite la direttiva <code>--nodes</code>
<code>--ntasks</code>	I task corrispondono al numero di processi MPI e il loro numero vengono specificato tramite questa direttiva
<code>--ntasks-per-node</code>	Offre la possibilità di controllare il numero di task per singolo nodo
<code>--cpus-per-task</code>	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 singolo nodo di computazione.

..... *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 $16 \cdot 4 = 64$ 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.

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/xxxxxxxx
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 precedente 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 <i>Mc</i> indica la memoria per core in MB, se ha suffisso <i>Mn</i> 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 forzatamente 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*:

<code>module list</code>	Elenca i moduli caricati
<code>module avail</code>	Elenca i moduli disponibili
<code>module load {nome-modulo}</code>	Carica un modulo
<code>module unload {nome-modulo}</code>	Rimuove un modulo
<code>module purge</code>	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.

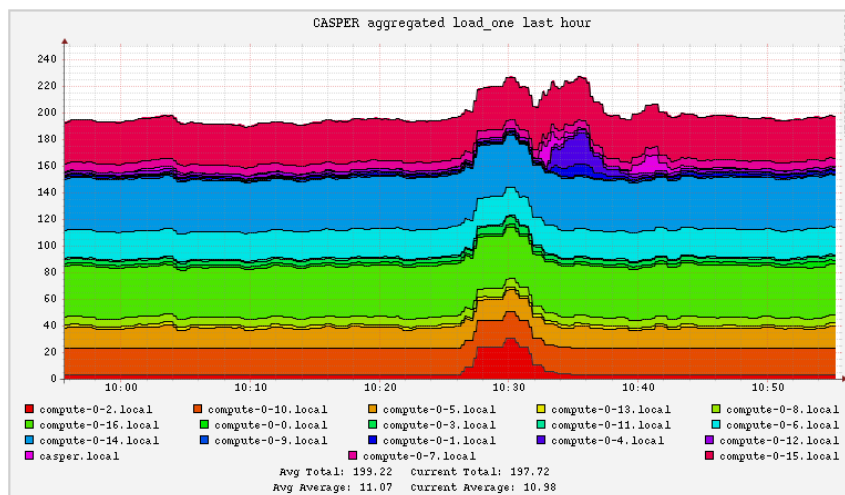


Figura 3: Ganglia - Ganglia example

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/>

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 \
-machinofile machinefile -mpi intel -mpiflags "-bootstrap slurm" \
-fabric ibv -fabricverbose -batch-report -batch run $SLURM_JOB_NAME.sim
```

.....*LAMMPS* [*lammmps.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 lammmps/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 possibilità 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 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 -s /bin/bash --nv tensorflowCentos.img bash -c 'python helloTensorflow.py'
```

Ora è possibile sottomettere il job con il comando sbatch:

```
sbatch singularity.sbatch
```