Guida al cluster di calcolo OPH – v1.0

24 novembre 2020

Sommario

Questo documento vuole essere semplicemente una guida base all'uso del cluster di calcolo del DIFA. Non può e non vuole essere una guida comprensiva né all'ambiente Linux, né al workload manager SLURM.

Viene data per acquisita una minima esperienza (da utente) con l'ambiente Linux e la capacità di creare codice per elaborare dati in maniera non interattiva (batch).

Le segnalazioni di problemi vanno indirizzate esclusivamente alla casella difa.csi@unibo.it che viene monitorata durante tutto l'orario di servizio. In questo modo si avrà risposta tempestivamente.

Capitolo 1

Generalità

Il DIFA dispone di una infrastruttura di calcolo parallelo utilizzabile dai gruppi di ricerca. Questa si compone di due distinti gruppi di nodi di calcolo con caratteristiche e modalità di utilizzo differenti (vedi Cap. 2):

- **BladeRunner:** gruppo dedicato principalmente a processi a basso parallelismo (`blade`, nodi di calcolo con hostname `bl0-NN`)
- **Matrix:** gruppo dedicato a processi massicciamente paralleli (`matrix`, nodi di calcolo con hostname `mtx-NN`).

L'accesso ad entrambi i gruppi di nodi avviene attraverso un server di frontend (IP statico: `137.204.50.71`, hostname: `str957-cluster`) con cui è possibile collegarsi e trasferire dati da macchine locali utilizzando i protocolli `ssh, scp, rsync`, etc. (vedi Cap. 3).

L'esecuzione di un qualsiasi codice (seriale o parallelo) sui nodi di calcolo avviene tramite un sistema di code gestito da uno specifico software di WorkLoad Management (SLURM, vedi Cap. 5): gli utenti inviano una richiesta di esecuzione del proprio codice che viene gestita dal WorkLoad Manager in base alla disponibilità delle risorse di calcolo.

Pertanto, la modalità tipica di utilizzo della infrastruttura di calcolo è la seguente:

1. l'utente si collega al frontend per preparare il proprio job (compilare l'eseguibile, trasferire i dataset, ecc);
2. in seguito, l'utente definisce le risorse necessarie all'esecuzione del proprio codice (tempo, numero di nodi/CPU, RAM) e richiede a SLURM (utilizzando una serie di comandi specifici di questo software, vedi Cap. 5) di inserire il job in coda;

3. una volta che il processo sia in coda, SLURM si occupa di schedularne l'esecuzione considerando:

    - i vincoli imposti dall'utente (le risorse richieste)
    - i vincoli di sistema (massimo tempo di esecuzione, eventuali down programmati)
    - la priorità della richiesta (che dipende da molti fattori, tra i quali il tempo di permanenza in coda e la quota di risorse assegnate già utilizzata dal gruppo a cui appartiene il richiedente)

È pertanto molto importante stimare al meglio le richieste di risorse per evitare da un lato che il job non riesca a terminare (se si è stati troppo stretti) e dall'altro che ne sprechi (se si è stati troppo larghi): le risorse richieste vanno a pesare sulla quota del gruppo di appartenenza e un utente che esageri sistematicamente (p.e. richiedendo più CPU o più memoria di quanto effettivamente usato dal codice) va a penalizzare tutto il gruppo.[1]

Per aiutare a stimare meglio numero di CPU e RAM da richiedere si può usare lo script `/home/software/utils/seff`, che va eseguito passando come unico argomento l'ID di un job terminato e che mostra in output la massima efficienza di uso delle CPU e della RAM: se l'efficienza indicata è bassa, si potevano richiedere meno risorse ottenendo lo stesso risultato. Soprattutto per la RAM si vedono spesso richieste eccessive anche di 3 ordini di grandezza.

[1] La richiesta di RAM viene 'convertita' in unità di CPU, quindi se si richiede tutta la RAM del nodo è come richiederne anche tutte le CPU, anche se il job potrà usare esclusivamente quelle richieste.

Capitolo 2

Risorse

2.1 Calcolo

Un cluster di calcolo parallelo si compone di una serie di *nodi* di calcolo collegati tra loro da una rete che può essere semplice Ethernet o una rete a bassa latenza come InfiniBand, che consente lo scambio di dati tra processori fisicamente e logicamente distinti.

Nel caso specifico del cluster DIFA, in ogni nodo ci sono generalmente due *socket* (S), ognuno con un certo numero di *cores* (C), che rappresentano le unità di calcolo fisiche del sistema. Ogni core può ospitare uno o due *thread* (T) di un processo, che rappresentano dunque due diverse unità di calcolo *virtuali* associate ad una unica unità di calcolo fisica. Questa seconda architettura si definisce *hyperthreading*. Pertanto, per ogni nodo di calcolo dotato di *hyperthreading* il numero totale di CPU virtuali (vCPU) è sempre il doppio rispetto al numero di *cores* fisici.

Da notare che i due thread di un core possono essere assegnati ad un solo processo, mentre core diversi di uno stesso socket possono essere assegnati a processi diversi. Tecnicamente, i thread condividono lo spazio di indirizzamento, i processi no. Per questo motivo, richiedendo una sola vCPU su un nodo con hyperthreading ne verranno comunque occupate (e addebitate) due, in quanto il secondo thread non potrebbe comunque essere usato da un diverso processo (vedi Cap. 4).

La tabella 2.1 riassume le caratteristiche hardware di tutti i nodi di calcolo di entrambi i gruppi.

2.2 Storage

L'infrastruttura di calcolo è dotata di due diverse aree di storage su disco con capienza di 42 TB e 240 TB, destinate ad ospitare le home folders degli utenti e

Gruppo (partizione)	hostname (str957-*)	#vCPU @ speed	Layout SxCxT	RAM (GB)	IB
BladeRunner					
b1	bl0-01	24 @ 2GHz	2x6x2	64	✓
b1	bl0-02	24 @ 2GHz	2x6x2	64	✓
b2	bl0-03	32 @ 2.4GHz	2x8x2	64	✓
b2	bl0-04	32 @ 2.4GHz	2x8x2	64	✓
b3	bl0-05	32 @ 2.1GHz	2x8x2	128	✓
b4	bl0-15	16 @ 2.27GHz	2x4x2	24	✓
b4	bl0-16	16 @ 2.27GHz	2x4x2	24	✓
b5	bl0-17	12 @ 2.4GHz	2x6x1	16	
b5	bl0-18	12 @ 2.4GHz	2x6x1	16	
b6	bl0-19	32 @ 2.2GHz	2x8x2	64	
b6	bl0-20	32 @ 2.2GHz	2x8x2	64	
Matrix					
m1	mtx-[00-15]	56 @ 2.2GHz	2x14x2	256	✓

Tabella 2.1: Struttura hardware del cluster DIFA, gruppi BladeRunner e Matrix. La prima colonna indica il gruppo e la seconda l'hostname dei diversi nodi di calcolo di cui si compone ciascun gruppo; la terza riporta il numero di CPU virtuali e la frequenza di clock dei processori di ciascun nodo; la quarta colonna descrive la struttura interna dei nodi in termini di numero di Socket (S), numero di cores (C), e numero di threads (T); la quinta colonna indica la memoria totale (RAM) del nodo di calcolo, mentre l'ultima colonna indica se il nodo è connesso con rete a bassa latenza InfiniBand (necessaria per le applicazioni parallele a memoria distribuita).

i dati prodotti o analizzati dai nodi di calcolo. Al fine di ottimizzare l'uso delle risorse di storage, lo spazio a disposizione è suddiviso in diverse aree (`home`, `home/web`, `home/work`, `scratch`) destinate ad usi diversi che ne determinano i vincoli e le modalità di utilizzo. L'accesso ai dati salvati nelle diverse aree può avvenire sia attraverso il frontend che attraverso i nodi di calcolo. In particolare, l'accesso tramite i nodi di calcolo ha velocità marcatamente diverse a seconda dell'area a cui si intende accedere:

Area `home`: Tutti i nodi accedono all'area `/home` via NFS. L'accesso è relativamente lento poiché avviene via Ethernet a 1GB (il server ha 2 interfacce, ma deve gestire il traffico di tutti i nodi). Lo spazio per la home è anche abbastanza limitato, il che implica la necessità di definire quote individuali molto stringenti per l'uso di questa area (vedi 2.2.1).

Area `scratch`: tutti i nodi accedono anche all'area `/scratch` . Questa è molto più veloce[1] e capiente della `/home` e le quote di utilizzo sono definite in base ai gruppi di ricerca di appartenenza di ciascun utente. Va sempre tenuto presente che si tratta di spazio "**prezioso**", da non usare per archiviazione a lungo termine: serve per conservare i dati da analizzare e i risultati delle elaborazioni fintanto che sono necessari. Poi vanno cancellati o spostati altrove.

2.2.1 Quota

A differenza della quota di calcolo, la quota disco una volta esaurita **impedisce di lavorare**: il disco è considerato pieno per tutto il gruppo e nessuno del gruppo over-quota riuscirà a lavorare sul "disco pieno" fin quando non verrà liberato spazio. Se si esaurisce la quota nella propria home potrebbe essere impossibile effettuare il login!

Vengono gestite quote diverse per diversi usi:

`/home/DOMINIO/nome.cognome`
: Dati personali, codici sorgente, dati non riproducibili, etc. Quote individuali di 50 GB per ciascun utente del cluster, indipendentemente dal gruppo di appartenenza.

`/home/web`
: Dati esportati via web, senza quota ma pubblici, accessibili tramite browser all'indirizzo
https://apps.difa.unibo.it/files/people/Str957-cluster/

`/home/software`
: Software messo a disposizione di tutti gli altri utenti del cluster; conteggiato come se fosse nella propria home

[1] Accesso tramite GlusterFS su IB a 100Gbps per i nodi in Matrix, a 40GBps i nodi in BladeRunner e il frontend, a 1GBps condiviso con la home per i nodi senza IB.

`/home/work`
: Dati per elaborazioni seriali o a basso parallelismo; quota per settore

`/scratch`
: Dati per elaborazioni massicciamente parallele; quota per settore

Per visualizzare lo spazio utilizzato nella home, usare il comando `quota`: se il primo numero è seguito da un asterisco, si è superata la propria soft-quota e si rischia l'errore di "disco pieno"!

Per visualizzare lo spazio utilizzato su `/scratch` è invece necessario usare il comando `df /scratch/settore`.

2.2.2 Data transfer

Per trasferire dati dal/al cluster si possono usare diversi strumenti.

Per piccoli trasferimenti (indicativamente **non più di 1G**) conviene usare `scp` o `rsync` sul frontend.

Per trasferimenti più grandi, invece, va creato un apposito job come se si trattasse di una normale elaborazione (richiedendo un solo processore e stimando il tempo necessario). Poi lo si accoda normalmente.

È prevista la creazione di uno o più "transfer node" dedicati proprio allo spostamento di file da/per il cluster, ma al momento non sono ancora attivi.

2.3 Software

Dal frontend è possibile accedere a diversi tool, in particolare compilatori C, C++, Fortran (vari dialetti) ed R.

Sui nodi di calcolo sono presenti solo gli ambienti runtime. Quindi attualmente non si possono usare i nodi per compilare. Se ci fosse richiesta potrebbero essere previsti nodi per job di compilazione.

Per quanto riguarda l'installazione di nuovo software, la regola è che a livello di sistema viene installato **solo ciò che è pacchettizzato da Debian nei repository ufficiali**. Tutto il resto va installato nella home. O nella propria (se non serve ad altri utenti) o sotto `/home/software` (in caso contrario). Anche ciò che viene messo sotto `/home/software` "pesa" sulla quota come se fosse nella propria home.

Il metodo più semplice (e standard) per i sorgenti che prevedono "`configure / make / make install`" è passare un opportuno parametro `--prefix` al comando `configure`.

Va ricordato che il frontend non è particolarmente potente ed è condiviso da molti utenti, pertanto deve essere usato solo per l'accesso al cluster e per la preparazione dei job, o per compilazione di codici di breve durata: eventuali compilazioni

lunghe vanno limitate (con nice e/o riducendo il numero di job paralleli) così da lasciare risorse disponibili anche agli altri. **Eventuali processi eccessivamente lunghi e resource-intensive in esecuzione sul frontend verranno terminati senza ulteriore preavviso**.

Capitolo 3

Accesso

Tutto il personale strutturato del DIFA ha la possibilità di accedere alle risorse di calcolo e storage dell'infrastruttura. A seconda degli accordi all'interno dei vari gruppi di ricerca, l'attivazione può essere automatica o necessitare di richiesta al proprio referente.

Il personale non strutturato può fare richiesta (motivata) al proprio tutor/relatore/ecc... che provvederà a fare richiesta al referente del gruppo.

La home di ciascun utente viene creata automaticamente al momento del primo accesso.

Sarà compito del referente del gruppo segnalare, a disattivazione avvenuta, quali sono le home da cancellare. **Le home di utenti disattivati e non accedute da più di 6 mesi potranno essere cancellate senza preavviso e senza possibilità di recuperare i dati**.

Gli attuali referenti calcolo di ciascun settore sono:

- `applicata`: **Claudia Sala, claudia.sala3@unibo.it**
- `astro`: **Marco Baldi, marco.baldi5@unibo.it**
- `atmos`: **Paolo Ruggieri, paolo.ruggieri2@unibo.it**
- `didattica`: **Olivia Levrini, olivia.levrini2@unibo.it**
- `materia`: **Cesare Franchini, cesare.franchini2@unibo.it**
- `nucleare`: **Angelo Carbone, angelo.carbone@unibo.it**
- `teorica`: **Elisa Ercolessi, elisa.ercolessi@unibo.it**
- `terra`: **Filippo Zaniboni, filippo..zaniboni@unibo.it**
- `esterni (INFN)`: **Daniele Cesini, daniele.cesini3@unibo.it**

3.1 Login

Per accedere alle risorse è necessario collegarsi in remoto al nodo di frontend (hostname: `str957-cluster`, IP statico: `137.204.50.71`) attraverso il protocollo `ssh` usando le proprie credenziali istituzionali (mail UniBO e relativa password). In alternativa alla mail completa, è possibile usare solo la parte nome.cognome (per il personale) o STUDENTI#nome.cognome (per gli studenti).

Ad esempio, un utente strutturato con indirizzo di posta elettronica istituzionale `nome.cognome4@unibo.it` potrà collegarsi al frontend con il comando:

```
ssh nome.cognome4@unibo.it@137.204.50.71
```

oppure con il comando:

```
ssh nome.cognome4@137.204.50.71
```

utilizzando la propria password istituzionale.

Uno studente con indirizzo di posta elettronica istituzionale `nome.cognome5@studio.unibo.it` potrà collegarsi al frontend con il comando:

```
ssh nome.cognome5@studio.unibo.it@137.204.50.71
```

oppure con il comando:

```
ssh STUDENTI#nome.cognome5@137.204.50.71.
```

3.1.1 Client Linux/Mac

Per gli utenti Linux e Mac non è necessario installare nulla: il comando ssh è già disponibile nel sistema.

Si consiglia di aggiungere al file `/.ssh/config` una sezione tipo:

```
Host cluster
    Hostname                    137.204.50.71
    User                        nome.cognome4@unibo.it
    Protocol                    2
    PasswordAuthentication      yes
    IdentitiesOnly              no
    ForwardX11                  yes
    ServerAliveInterval         30
    ServerAliveCountMax         5
    TCPKeepAlive                yes
```

Sarà così sufficiente usare:

```
ssh cluster
```

Se non si intendono utilizzare sul frontend applicativi che richiedano X11, é più sicuro impostare ForwardX11 a "no".

3.1.2 Client Windows

Agli utenti Windows si raccomanda l'uso di MobaXTerm come software di shell emulator invece di Putty, soprattutto se necessitano di usare tool grafici (gedit, matlab, ...).

Capitolo 4

Accounting

Ogni utente fa capo ad uno o più account. L'account di default per ciascun utente è quello del settore di appartenenza all'interno del DIFA, mentre eventuali altri account (legati a specifici progetti di ricerca o attività didattiche) devono essere specificati al momento della richiesta di risorse attraverso il WorkLoad Manager.

Le risorse usate da un job vengono "addebitate" all'account di appartenenza indicato al momento della sottomissione del job stesso.

Le risorse di calcolo utilizzate da ciascun account (settore) influenzano la priorità dei job: maggiore è la quota di risorse già usata dagli utenti di un certo settore, minore sarà la priorità assegnata ai job di quel settore. In questo modo, quando i nodi con le risorse richieste non vengono usati da altri utenti, anche chi ha esaurito la propria quota di risorse può continuare a lavorare. Allo stesso tempo, chi invece non l'ha esaurita non dovrà aspettare troppo a lungo per veder partire i propri job nel caso in cui il cluster sia completamente occupato da altri job (il cosiddetto fair share delle risorse).

Il calcolo della quota fairshare è unico per tutto il cluster ma il calcolo dell'utilizzo delle risorse è diverso a seconda dei nodi utilizzati: per questo è bene separare la fase di elaborazione (generalmente parallela) da quella di postprocessing (generalmente seriale).

Capitolo 5

Gestione dei job

Il nodo di frontend ha la sola funzione di consentire l'accesso da remoto ai cluster di calcolo e di editare/compilare i codici sorgente. Pertanto, non deve **mai** essere utilizzato per eseguire codici che facciano uso intensivo di risorse di calcolo in quanto questi creerebbero inevitabilmente problemi a tutti gli altri utenti fino a rendere impossibile l'accesso all'intera infrastruttura. Nel caso sia necessario effettuare dei test di esecuzione di un codice prima di sottometterlo al WorkLoad Manager, ogni utente dovrà accertarsi (**attivamente**) che tale test non duri più di qualche secondo. È raccomandato l'uso della partizione di debug per i test che richiedano da 1 a 15 minuti.

5.1 Sottomissione

Per eseguire un codice seriale o parallelo è necessario utilizzare il WorkLoad Manager SLURM che si occuperà di allocare le necessarie risorse e di gestire la priorità delle richieste.

Per ogni richiesta di esecuzione (job) è dunque necessario dichiarare (attraverso un batch script, vedi sotto) le risorse necessarie (numero di nodi, numero di processori, memoria, tempo di esecuzione) ed eventuali constraint (gruppo di nodi, presenza di InfiniBand, tipo di CPU). Facoltativamente, possono essere indicati anche altri parametri (eventuale account secondario sul quale addebitare l'utilizzo delle risorse, indirizzo mail dove inviare le segnalazioni relative allo stato del job, ecc...).

Benché sia possibile fornire tutte queste informazioni al WorkLoad Manager attraverso parametri a linea di comando, normalmente si preferisce creare un file di shell detto job script così da evitare di dover riscrivere tutto ogni volta. Il file dovrà contenere, oltre alle istruzioni di lancio del job vero e proprio, delle righe che inizino con la stringa `#SBATCH` seguita dall'opzione che si vuole specificare (ad

esempio il numero di nodi da allocare, il numero di processori per nodo, la memoria, il tempo di esecuzione, ecc.). Se un'opzione viene specificata sia nel file che nella command line, il valore dato da command line ha la priorità su quello indicato nel file.

Un esempio di job script, che supponiamo di chiamare `nomefile.sh`, con alcune opzioni i cui valori possono essere modificati in base alle esigenze di ciascuna esecuzione, è dato dalla struttura seguente:

```
#!/bin/bash
#SBATCH --account=formazione
#SBATCH --time=0:00:30
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --constraint=blade,ib  # Per MPI meglio avere InfiniBand,
#                               e i nodi di Blade sono sufficienti
#SBATCH --mem=2000 # Memory per node (in MB).
#SBATCH --job-name="test constraint"
#SBATCH --output=a%j.out
#SBATCH --mail-user=nome.cognome4@unibo.it
#SBATCH --mail-type=ALL
cd ${HOME}/path/to/exec
mpirun ./a.out otherparams
```

Per inviare la richiesta di risorse definita da questo job script al WorkLoad Manager si dovrà eseguire il comando:

```
sbatch nomefile.sh [altri argomenti]
```

Nota: la parte opzionale 'altri argomenti' sarà disponibile al job script come se venisse eseguito sul frontend (quindi $1, $2, ecc.). Se un'elaborazione va lanciata con un argomento diverso per 10 volte, quindi, si può creare un solo job script e passare di volta in volta un diverso valore sulla riga di sbatch.

Per ulteriori informazioni sulle opzioni del WorkLoad Manager SLURM è possibile fare riferimento al manuale che si può consultare attraverso il comando `man sbatch` o alla documentazione online.

Se il job non utilizza parallelizzazione MPI o richiede un solo nodo, va omesso mpirun e indicato direttamente l'eseguibile.

È possibile usare più job step (più righe che lancino eseguibili come mpirun) in un unico jobfile se ogni step richiede **la stessa allocazione di risorse** del precedente e deve partire quando il precedente è terminato. Se invece gli step sono **indipendenti o sequenzialmente dipendenti con diverse richieste di**

risorse, allora conviene usare job script separati: l'esecuzione dei jobstep avviene sequenzialmente all'interno di un'unica allocazione di risorse (p.e. in un unico sottoinsieme di nodi), mentre job diversi possono avere allocazioni diverse (quindi riducendo lo spreco di risorse) e partire anche parallelamente. Si rimanda alla documentazione di sbatch per definire le dipendenze tra job (after, afterany, ...).

Visto che il cluster è particolarmente eterogeneo (ovvero contiene nodi di calcolo diversi tra loro per numero di core e memoria disponibile) è sempre opportuno specificare bene le necessità del job. I constraint attualmente definiti sono `blade` (nodi per postprocessing e job a basso parallelismo), `matrix` (job ad elevato parallelismo), `ib` (nodi con InfiniBand), `avx` (nodi con supporto istruzioni AVX), `intel` (nodi con processori Intel) e `amd` (nodi con processori AMD). È fortemente sconsigliato, in generale, richiedere un nodo specifico o una partizione specifica, a meno che non si stiano facendo test comparativi (p.e. benchmarking) che lo richiedono.

Un job che richieda più di un nodo riceverà automaticamente nodi **omogenei**. Questo, in blade, limita ad usare al più 2 nodi. Se il job può lavorare bene anche con nodi eterogenei (p.e. perché i processi sono quasi indipendenti), è possibile creare un **heterogeneus job**[1] specificando le diverse partizioni (o perfino i singoli nodi) utilizzabili dalle diverse componenti del job.

Se non viene specificato diversamente (usando l'opzione `--account`), le risorse utilizzate vengono addebitate all'account di default dell'utente che sottomette il job script, ovvero al budget di risorse del suo settore di appartenenza. Per semplicità e chiarezza è sempre meglio specificare nello script l'account da usare.

Per evitare di impattare i job di altri utenti con job "misbehaving", le risorse disponibili vengono limitate a quelle richieste, anche se sul nodo ce ne fossero disponibili altre libere. In particolare, se viene richiesto un solo processore, e nel nodo allocato ne rimangono 31 non utilizzati, il job può comunque usare esclusivamente il solo processore richiesto. Se il job tenta di generare più task, questi gireranno comunque su un solo processore.

Da notare che la RAM è una risorsa controllata. Se non viene specificato diversamente, verranno assegnati 200MB per ogni CPU richiesta (che sono volutamente pochi, quindi è meglio specificare sempre l'opzione `--mem` nel job script!

Attenzione a quanta RAM viene richiesta! Soprattutto provando a richiedere "*tutta quella disponibile*" il job potrebbe non partire: una parte della RAM è riservata per l'uso da parte dell'HW, ed un'altra parte per i programmi di sistema. Normalmente queste "riserve" sono di circa 3GB. Il valore **massimo richiedibile** su un nodo è RealMemory - MemSpecLimit (entrambi i valori si possono vedere usando `scontrol show node str957-...`).

[1] Vedi https://slurm.schedmd.com/archive/slurm-18.08.0/heterogeneous_jobs.html

5.2 Gestione

Una volta che un job è stato inviato al WorkLoad Manager tramite il comando `sbatch` è possibile monitorare lo stato di priorità e di avanzamento del job con una serie di comandi di gestione:

- `slurmtop` visualizza lo stato del cluster in modo "semigrafico"
- `squeue` visualizza lo stato delle code
- `scancel <job-ID>` per annullare l'esecuzione di un job
- `scontrol show job <job-ID>` per vedere i dettagli di un job (compresa la priorità accumulata in coda)
- `/home/software/utils/seff <job-ID>` per vedere con quanta efficienza sono state utilizzate le risorse richieste. Il job deve già essere terminato.
- `sshare` e `/home/software/utils/showfullusage.sh` per vedere quante risorse sono già state utilizzate e da chi.

Capitolo 6

Approfondimenti

http://slurm.schedmd.com/tutorials.html
http://slurm.schedmd.com/documentation.html