Cenni su LSF

Utilizzo dell'Infrastruttura Cresco

Ing. Fiorenzo Ambrosino, PhD
DTE-ICT-HPC

Frascati – 5 Aprile 2017

- *Informazioni utili su CRESCO*
- *Comandi Base di LSF*
  - *Code, sottomissione, monitoring*
- *Esempi e casi particolari*
  - ✓ *Job MPI*
  - ✓ *Flussi di Job*
  - ✓ *Job Array*

Informazioni utili su CRESCO

Link utili:

- Portale web CRESCO: http://www.cresco.enea.it/
- Faro2: http://www.cresco.enea.it/nx-2.php
- Documentazione sui sistemi Cresco: http://www.afs.enea.it/project/eneagrid/Resources/CRESCO_documents/index.html
- Documentazione ENEAGRID: http://www.eneagrid.enea.it/
- Primo accesso & HowTo: http://www.cresco.enea.it/helpdesk.php
- JobRAMA Job Monitor: https://jobrama.enea.it/
- ENEAGRID Software: https://gridsrv.enea.it/egrid_software
- Grid Ticket System: https://gridticket.enea.it/

LSF – Le Code

Code

I job vengono sottomessi tramite le code;

Le code sono dei contenitori di job che hanno certe regole in base alle quali i job vengono schedulati;

Con la coda vengono definiti alcuni parametri tra i quali:

- Utenti abilitati;
- Massimo tempo di durata del job (runlimit);
- Regole sul numero di core (minimo, massimo, multiplo di);
- Priorità;
- Quali e quanti nodi di calcolo possono essere occupati dalla coda;
- Quanti core possono essere in running contemporaneamente;

LSF – bqueues

bqueues

Il comando mostra l'elenco delle code definite sul site LSF;

```
~@cresco4-fg1.portici.enea.it>bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
cresco4_256h24    24  Open:Active      -    -    -    - 29120 27616   560     0
cresco4_16h24     24  Open:Active    256    -    -    -   640   416   224     0
cresco4open_256   22  Open:Active      -    -    -    -  1568  1568     0     0
cresco4_h144      22  Open:Active     47    -    -    -    86    39    46     0
cresco4_h6        22  Open:Active      -    -    -    -   576     0   576     0
cresco3_72h24     20  Open:Active      -    -    -    -  3552  2640   792     0
cresco3_h144      18  Open:Active      -    -    -    -   346    10   330     0
cresco5_16h24     10  Open:Active      -    -    -    -   400     0   400     0
cresco5_h144      10  Open:Active      -    -    -    -   144   112    32     0
cresco4test_h14  100  Open:Active      -    -    -    -     0     0     0     0
system           100  Open:Active      -    -    -    -     0     0     0     0
small_10m         30  Open:Active      -    -    -    -     0     0     0     0
small_h144         3  Open:Active      -    -    -    -   310    86   192     0
cresco5_h3         1  Open:Active      -    -    -    -     0     0     0     0
...
```

LSF – bqueues

bqueues –l nome_coda

Il comando mostra dettagli sulla configurazione della coda;

```
~@cresco4-fg1.portici.enea.it>bqueues -l cresco4_16h24
QUEUE: cresco4_16h24
  -- Queue for small parallel jobs (max slots=256).  Featuring Intel Sandy Bridge (Xeon E5-2670) nodes.
PARAMETERS/STATISTICS
PRIO NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN SSUSP USUSP  RSV
 24    0  Open:Active     256    -    -    -   640   416   224     0     0    0
 RUNLIMIT
 1440.0 min of cresco3x002.portici.enea.it
 PROCLIMIT
 16 16 256
...
USERS: cresco4users/
HOSTS:  intel_E5_2670_large_job/
POST_EXEC:  /afs/enea.it/software/lsf/bin/dp_post_exec
RES_REQ:  select[type==large_job && cresco4 && node_up == up] span[ptile=16]
Maximum resource reservation time: 57800 seconds
```

LSF – sottomissione job

Per sottomettere un job su Cresco occorre in genere specificare:

- Sezione su cui sottomettere (coda);
- Numero di core richiesto;
- Eventuale utilizzo di risorse particolari;
- Eseguibile da lanciare con eventuali opzioni;

LSF prenderà in carico il job, gli assegnerà un ID e in base alle proprie policy di scheduling (priorità, fairshare, backfill, ecc) e disponibilità delle risorse richieste (numero di core richiesti e disponibili, durata prevista del job, eventuale memoria, precondizioni eventualmente impostate, ecc) gli assegnerà uno status.

```
~@cresco4-fg1.portici.enea.it>bsub sleep 10
Job <2522> is submitted to default queue <small_10m>
```

LSF – bsub

bsub

Il comando sottomette il job su Cresco

Alcune opzioni:

- -n specifica il numero di core
- -q specifica la coda
- -R specifica richiesta di risorse
- -W specifica una runlimit diversa da quella della coda
- -e file di error (standard error)
- -o file di otput (standard uotput)
- -w per job con dipendenze

Es:

bsub –n 64 –q cresco4_16h24 –e errorfile –o outputfile –W 30 mioscript.sh

LSF – job status

Un job può trovarsi in uno di 4 stati:

- Running
- Pending
- Done
- Exit

Il comando **bjobs** restituisce informazioni sullo stato dei propri job sottomessi

L'opzione **–l** restituisce informazioni aggiuntive come: stato del job, data di sottomissione, data di start (se c'è), data di stop (se c'è), coda, numero di core, risorse particolari richieste, runlimit ecc.

L'opzione **–p** (come anche –l), in caso di job in Pending, restituisce informazioni aggiuntive sulle motivazioni del Pending.

Opzione **–u** seguita da un nome utente restituisce i job sottomessi dall'utente specificato.

Il comando **bkill** termina un job che è in Running o in Pending

LSF – job monitoring

Comandi per monitorare lo stato di occupazione dei nodi di calcolo da parte di LSF:

lsload

Monitoring risorse dinamiche per host

lshosts

Monitoring le risorse statiche per host

bhosts

Mostra il numero di job sui vari host

lsxlosd

Unisce le informazioni degli altri comandi (necessita di righe da 132 caratteri)

LSF – freecresco

E’ stata sviluppata l'utility freecresco che permette di controllare gli slot liberi sulle varie code per sapere quanti core si possono usare e per quanto tempo, in modo da poter andare immediatamente in Running

```
~@cresco4-fg1.portici.enea.it>freecresco
```

Queue name	Specific resource option	Available slots throughout queue runtime limit	Available slots for backfill runs	Backfill time window (minutes)
cresco3_72h24	-	-	192(72,192,24)	336
cresco3_h144	-	-	3(1,3,1)	5690
			5(1,5,1)	5689
cresco4_256h24	-	336(256,336,16)	464(256,464,16)	1416
cresco4open_256h24	-	336(256,336,16)	464(256,464,16)	1416
cresco4_16h24	-	32(16,32,16)		
cresco4_h6	-	464(1,464,1)		
cresco5_h144	-	8(1,8,1)		
cresco5_16h24	-	32(16,32,16)		

```
See "freecresco -h" for documentation and known bugs.
```

Job MPI

```
/* hello.c
*
* Simple "Hello World" program in MPI.
* NOTA: Programma di test preso in prestito dal sito del CERN
*/

#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[]) {
int numprocs; /* Number of processors */
int procnum; /* Processor number */
/* Initialize MPI */
MPI_Init(&argc, &argv);
/* Find this processor number */
MPI_Comm_rank(MPI_COMM_WORLD, &procnum);
/* Find the number of processors */
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
printf ("Hello world! from processor %d out of %d\n", procnum, numprocs);
/* Shut down MPI */
MPI_Finalize();
return 0;
}
```

Job MPI

- Compilazione:

– mpicc hello.c

- Per lanciare il job, crearsi un semplice script di lancio (wrapper):

```
#!/bin/sh
PROCS=`cat $LSB_DJOB_HOSTFILE | wc -l`
mpirun -n $PROCS --mca pls_rsh_agent "blaunch.sh" –hostfile $LSB_DJOB_HOSTFILE ./a.out
```

- Lo script deve

– Contare i processori assegnati da LSF

– Sostituire eventualmente “ssh” con “blaunch.sh”

- Esempio di lancio del programma:

bsub -n 64 -q cresco_16h24 –W 10 -o test.%J ./lancio.sh

Job array

I job array sono utili quando bisogna lanciare uno stesso codice di calcolo seriale su un grande numero di dati diversi

```
bsub -J "myArray[1-1000]" myJob
Job <123> is submitted to default queue <normal>.
```

Nell'esempio vengono sottomessi 1000 job seriali dello script myJob

Lo script differenzia i vari casi in base alla variabile `$LSB_JOBINDEX`

Es:

```
$!/bin/sh
cd ./caso-$LSB_JOBINDEX
./solver.sh
```

Flussi di Job

- I flussi sono utili in molti ambiti:
  - Job dipendente dal file di output di un altro
  - Job che collezioni l'output di diversi job precedenti
  - Sequenze di job automatizzate (loop)

- Per pianificare un flusso di job, è necessario conoscere un po' di scripting, alcune nozioni di LSF ed un minimo di progettazione di algoritmi.

L'esempio applicativo sui flussi di job è stato preparato da A. Secco ed è reperibile sul sito www.cresco.enea.it

Flussi di Job

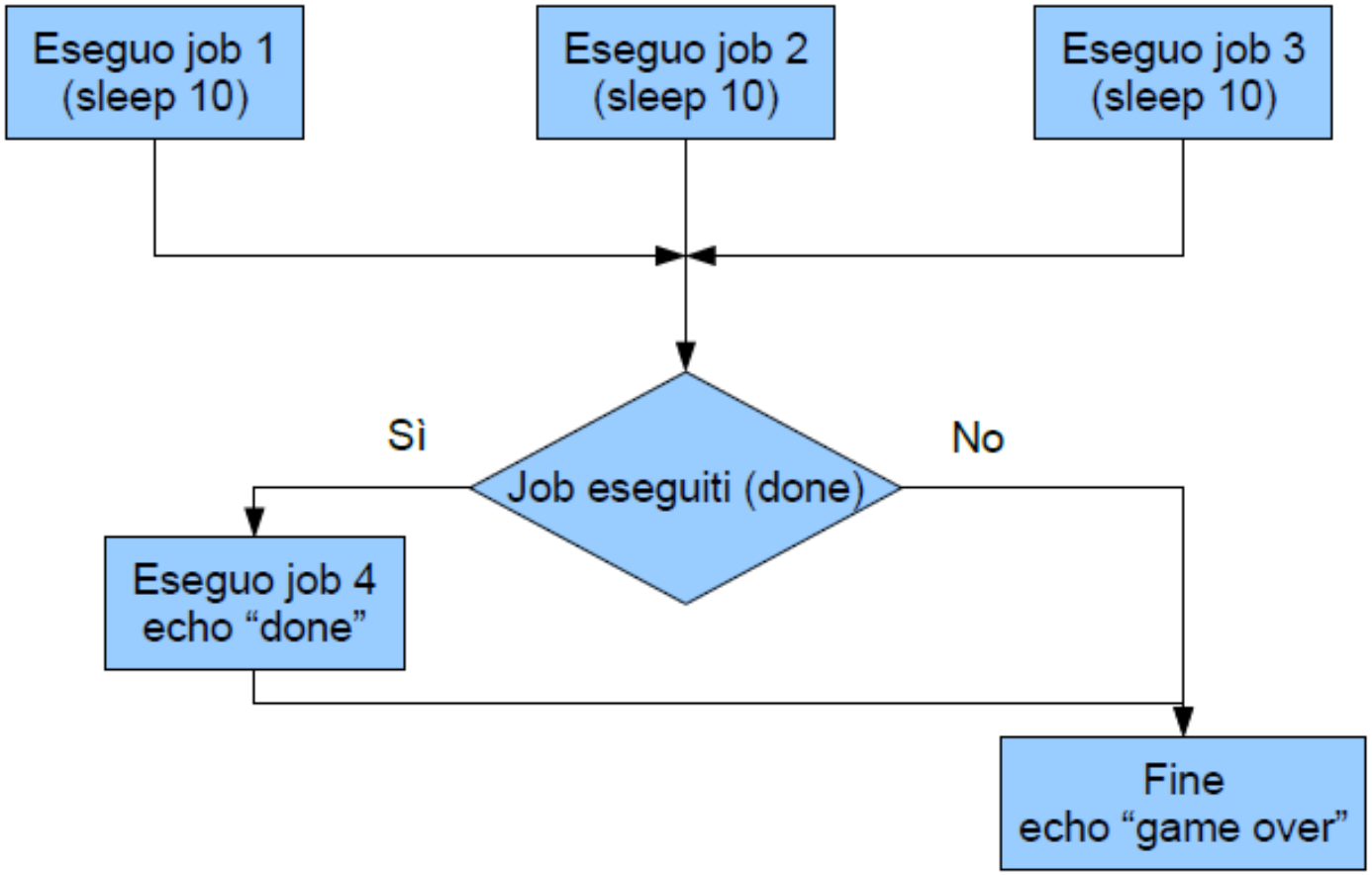
- ✓ In molti casi servirà avere esattamente il jobid come output di bsub:

```
bsub ... | awk -F'<' '{print $2}' | awk -F'>' '{print
$1}‘
```

- ✓ Si potrebbe mettere il filtro in uno script dentro la propria home directory (jobid.sh):

```
#!/bin/sh
awk -F'<' '{print $2}' | awk -F'>' '{print $1}'
```

Flussi di Job



Flussi di Job

```
sub.sh:
#!/bin/sh
JOBID1=`bsub sleep 10 | ./jobid.sh`
JOBID2=`bsub sleep 10 | ./jobid.sh`
JOBID3=`bsub sleep 10 | ./jobid.sh`
export JOBID1 JOBID2 JOBID3
DEPEND1=`echo "done($JOBID1) && done($JOBID2) && done($JOBID3)"`
JOBID4=`bsub -w "$DEPEND1" -o done ./done.sh | ./jobid.sh`
export JOBID4
DEPEND2=`echo "exit($JOBID1) || exit($JOBID2) || exit($JOBID3) ||
ended($JOBID4)"`
JOBID5=`bsub -w "$DEPEND2" -o clean ./clean.sh`

done.sh:
#!/bin/sh
echo “$JOBID1, $JOBID2, $JOBID3 done”

clean.sh:
bkill $JOBID1 $JOBID2 $JOBID3 $JOBID4 >/dev/null 2>&1
echo “game over”
```

Grazie per l'attenzione!