ENEA GRID

Corso di introduzione all'uso ed all'amministrazione

Autore: Alessandro Secco
alessandro.secco@nice-italy.com

Lezione 4

- Riepilogo lezione 3
- LSF: Job environment
- LSF: Lancio di un job multi-caso
- LSF: Introduzione al lancio di un job parallelo

Riepilogo lezione 3

- Nella scorsa lezione si sono introdotti alcuni semplici argomenti di amministrazione.
- AFS:
  - Repliche dei volumi
- LSF:
  - Aggiunta di un host
  - Aggiunta di una risorsa boolean
  - Aggiunta di una coda

LSF: Job environment

Job environment

- Una delle caratteristiche più importanti di LSF è che il job in esecuzione viene fatto girare in un environment il più possibile simile a quello di lancio.
- Il parametro -L <shell> di bsub permette di lanciare il job all'interno di una shell “pulita”, in cui l'environment non viene modificato.
- In alcuni casi è necessario lanciare i propri job da uno script che modifichi l'environment a seconda delle proprie esigenze.
- In molti casi invece lo script di lancio potrebbe utilizzare l'environment del job in modo produttivo.

Job environment

- LSF fornisce un environment aggiuntivo al processo in esecuzione, che può essere sfruttato per sviluppare script di lancio molto potenti.
- Si suggerisce un comando in grado di rivelare in modo semplice l'environment di LSF:
  - bsub -I env | grep LS

Alcune variabili interessanti

- **LSB_JOBID**: valore del jobid
- **LSB_JOBINDEX**: indice del job (multi-caso)
- **LSB_MCPU_HOSTS**: nome degli host selezionati e numero di cpu per ogni host
- **LSB_HOSTS**: nome degli host selezionati (lo stesso host viene ripetuto se girano più processi)
- **LSB_SUB_HOST**: host di sottomissione
- **LSB_QUEUE**: coda di sottomissione
- **LSB_JOBPID**: pid padre generato da LSF
- **LSB_JOBNAME**: nome (comando) del job lanciato
- **LSB_DJOB_HOSTFILE**: nome del file di host creato da LSF

Esempio di utilizzo

- Si crei un piccolo script “hello.sh”:

```
#!/bin/sh

echo “Ciao dal job $LSB_JOBID”
echo `hostname`
echo “LSB_HOSTS=$LSB_HOSTS”
```

- Si lanci lo script con **bsub -I ./hello.sh**, dopo averlo reso eseguibile.
  - Cosa visualizza in output ?
  - Cosa cambia se si aggiunge -n 2 a bsub ?

LSF: Lancio di un job multi-caso

Job multi-caso

- Un job multi caso è il lancio contemporaneo di un certo numero di job LSF dalla stessa linea di comando.
- Alcuni esempi:
  - Si deve lanciare lo stesso eseguibile su molti file di input
  - Si deve lanciare lo stesso eseguibile con molti parametri diversi
- Più in generale, si parla di multipli casi quando si può indicizzare il proprio lavoro, sfruttando la possibilità di eseguirne più parti contemporaneamente.

Job multi-caso

- In alcuni casi è possibile scomporre job seriali di grandi dimensioni in tanti piccoli job.
  - Un job seriale può girare solo su una macchina.
  - Tanti piccoli job possono sfruttare tanti processori contemporaneamente e, quindi, ottenere prestazioni di ordini di grandezza superiori
- **NOTA BENE**: è bene parlare con il proprio amministratore di sistema prima di lanciare un numero molto elevato di casi.

Job multi-caso

- Vantaggi:
  - Risultato in tempi più brevi
  - Controllo centralizzato dei job
  - Miglior utilizzo delle risorse di calcolo
- Svantaggi:
  - Può essere invasivo
  - Ci si deve porre il problema della scalabilità
  - Preparazione del job più difficoltosa

Esempio 1: hello world

- Nel primo esempio, da una directory vuota, si lancia lo script “hello.sh” per 30 volte per vedere l'output generato

```
#!/bin/sh

echo “Ciao dal caso $LSB_JOBINDEX del job $LSB_JOBID”
echo “mi trovo sull'host `hostname`”
```

- Si lanci ora il job:
  - bsub -J 'hello[1-30]' -o output.%J.%I ./hello.sh

-J dà un nome al job ed eventualmente definisce un indice

-o definisce il file di output. %J è il numero del job, %I l'indice

Esempio 2: sleep job

- Lanciare uno sleep job in multi-caso è utile per impratichirsi con i comandi di controllo forniti da LSF
- Si lanci e si commenti:
  - bsub -J 'sleep[1-30]' sleep 1000
  - bjobs
  - bjobs -l '<jobid>[3]'
  - bkill '<jobid>[1]'
  - bkill '<jobid>'

Esempio 3: script multipli

- Come lanciare 5 script che differiscono solo per il nome indicizzato ?
- launcher.sh:

```
#!/bin/sh

./caso.$LSB_JOBINDEX
```

- caso.1 (caso.2..5)

```
#!/bin/sh

echo “caso 1”
```

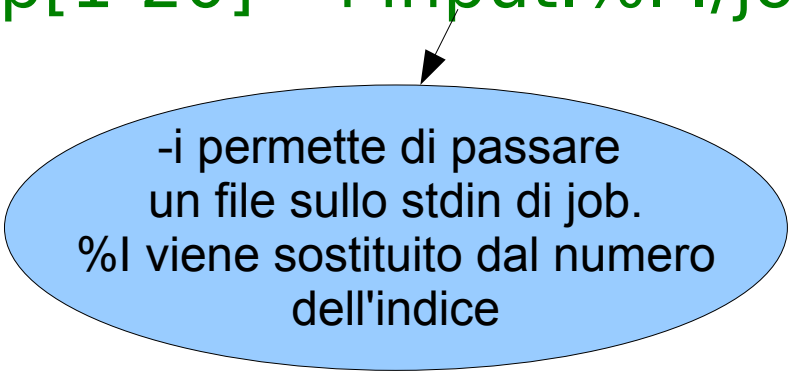
- Si lanci:
  - bsub -J 'multiscript[1-5]' ./launcher.sh

Esempio 4: matrice

- Un indice non basta... si possono avere 2 indici ?
- Sì. Supponiamo che la matrice sia 4x5.
  - Il numero di casi sarà 4x5=20 (supponiamo di lanciare bsub -J “test[1-20]” ...)
  - Il primo indice si calcola come modulo della divisione tra $LSB_JOBINDEX-1 e 4
  - Il secondo indice è la divisione intera tra $LSB_JOBINDEX-1 e 4
- È anche possibile avere indici a più di 2 dimensioni: provare come esercizio.

Esempio 5: file di input multipli

- Un modo per passare ad un eseguibile file di input multipli è quello di usare l'esempio 3: anziché richiamare n script diversi, si devono passare n file diversi.
- Se il job supporta gli input tramite stdin, i può provare con
  - bsub -J”multi_inp[1-20]” -i input.%I ./job



Introduzione al lancio di un job parallelo

Job multithreading

- I job multithreading sono stati definiti come job che generano più thread su di un singolo host.
- Questi non sono propriamente job paralleli, ma possono sfruttare l'hardware a più processori in modo molto efficiente.

Esempio 1: Job multithreading

- Si supponga di avere un job multithreading che venga lanciato in questo modo:
  - ./job -i <input_file> -x <threads>
- Il comando **bsub** ha l'opzione **-n <nprocs>** che dice di utilizzare nprocs processori, presi da diversi host.
- Il comando bsub necessita quindi di 2 informazioni:
  - Il numero di thread (1 thread = 1 processore)
  - Il job deve girare su un singolo host
- Soluzione:
  - bsub -n <threads> -R “span[hosts=1]” ./job -i <input_file> -x <threads>
- span[hosts=1] richiede che tutti i processori siano sullo stesso host.

Problema: ingannare LSF

- LSF, come tutti gli altri resource manager, non sostituisce il sistema operativo: *in alcuni casi può essere ingannato*.
- Si lanci per esempio il job di prima senza specificare -n <threads>: cosa cambia ?
  - bsub -R “span[hosts=1]” ./job -i <input_file> -x <threads>
- Nel file lsb.queues si può specificare il parametro THREADLIMIT, che può limitare questo problema.
- Politiche severe di amministrazione di sistema rimangono comunque il miglior deterrente ad usi scorretti.

Job paralleli: premessa

- Un job parallelo può essere lanciato su N macchine diverse e su un numero di processori diverso per ogni macchina.
- MPIch, come LamMPI ed altre implementazioni delle librerie MPI, usano strumenti esterni ad LSF per lanciare i processi remoti, come rshd, sshd o demoni propri.
- Verrà mostrato solamente il lancio semplice, dove LSF non si interpone tra mpich ed i processi.
  - VANTAGGI: semplicità di utilizzo
  - SVANTAGGI: LSF non darà informazioni di accounting sul job.

Job mpi per i test (hello.c)

```
/*  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 per i test

- Prepararsi l'ambiente
  - Per l'esercizio che segue si userà mpich:
    - PATH=/afs/enea.it/software/mpich/@sys/bin:$PATH
    - export PATH
  - Compilare il job (eseguire su aix e linux):
    - fs sysname (leggo il nome della piattaforma)
    - mkdir <nome>
    - mpicc ./hello.c -o @sys/hello
  - Testare il job (funziona solo sulle macchine in cui funziona rsh <nomehost> comando)
    - mpirun -np 4 @sys/hello

Esercizio: lancio di un job mpi

```
#!/bin/sh

P4_RSHCOMMAND=/afs/enea.it/software/lsf/conf/dev/bin/mpich/rsh
PATH=/afs/enea.it/software/mpich/@sys/bin:$PATH
export PATH P4_RSHCOMMAND

#BSUB -n 10
#BSUB -R '(type==LINUX || sp5) && mpich span[ptile=1]'
#BSUB -o job_mpich_%J.out
#BSUB -e job_mpich_%J.err

NUMPROC=`cat $LSB_DJOB_HOSTFILE|wc -l`
mpirun -np $NUMPROC -machinefile $LSB_DJOB_HOSTFILE @sys/hello
```

- Lanciare con
  - bsub < script.sh
- Cosa succede ?

Esercizio: analisi

- **P4_RSHCOMMAND**: è una variabile di ambiente letta da mpirun che sovrascrive il comando rsh con uno script che manda, tramite LSF, il token di AFS su tutti i nodi
- **span[ptile=1]**: dice ad LSF di mandare un processo per nodo.
- L'opzione **-R** di bsub può ospitare all'interno diverse funzioni: in questo caso si richiede una selezione di risorse ed uno span.
- Per lo stesso job i processi possono partire su due piattaforme diverse tra loro !

Problemi

- P4_RSHCOMMAND sostituisce rsh con uno script di LSF per portare il token su tutti i nodi. Purtroppo:
  - non è una soluzione scalabile
  - potrebbe avere problemi di affidabilità
- Attualmente si sta lavorando a soluzioni più affidabili che comunque non modificheranno quanto mostrato nell'esercizio.
- Per il lavoro di tutti i giorni, si suggerisce di usare gli script preparati dagli amministratori, come mostrato in seguito.

Lancio dei job MPI

- Per lanciare job mpi, il team di Enea grid fornisce alcuni script utili che automaticamente preparano l'ambiente in cui andranno a girare i job.
- Questi script funzioneranno in una buona parte di casi, specialmente per chi ha i sorgenti dei propri job.
- Gli script sono già integrati con AFS, hanno un help, sono supportati, e le loro modifiche sono trasparenti all'utente.
- Dove non esistesse uno script, è consigliato parlare con il proprio amministratore prima di iniziare con il proprio lavoro.

Script di lancio di job mpi

- mpich.gen, lancio di job mpich:
  - mpich.gen @sys/hello -np 2 -sp5 -queue small_10m
- poe.sub, lancio di job mpi legati alla piattaforma IBM AIX (comprende l'uso degli switch ad alte prestazioni tra nodo e nodo).
  - poe.sub ./a.out -procs 2 -sp5 -queue small_10m
- Chiedere agli amministratori per il supporto lammpi, mpich2, openMP.
- Tramite script diversi, sono supportati anche codici commerciali e non, come mcnp, mcnpx, fluent, abaqus, ecc.

Informazioni per gli amministratori

- La documentazione ufficiale su come integrare LSF e job paralleli si trova su:
  - /afs/enea.it/software/lsf/docs/lsf/7.0.2/lsf_hpc_using/index.html
- Purtroppo questa documentazione non comprende l'integrazione AFS, che sarà oggetto di un'altra lezione
- Per un corretto funzionamento dei job paralleli è necessario evitare che ci siano macchine in stato “-ok” (comando lsload)