Manuale utente del cluster Linux

Giancarlo Bartoli (giancarlo.bartoli@isti.cnr.it)
Tiziano Fagni (tiziano.fagni@isti.cnr.it)

29 marzo 2004

In questo rapporto tecnico descriveremo come poter usare, in modo proficuo, il cluster Linux gestito dal gruppo di ricerca HPC (High Performance Computing) dell'istituto ISTI presso il CNR di Pisa.
Inizialmente faremo una panoramica dell'architettura del sistema descrivendo le caratteristiche principali dei nodi e i meccanismi utilizzati per permettere a questi ultimi di interagire all'interno del sistema. Successivamente saranno descritte in dettaglio le modalità operative per accedere e muoversi all'interno del cluster. Per ultimo, infine, descriveremo alcune operazioni tipiche che un utente puó eseguire su un sistema di questo tipo.

1 Architettura del cluster

Il cluster è composto da 9 nodi, ognuno avente le seguenti caratteristiche:

- scheda madre Intel SE7500WV2 con doppio processore Intel Xeon 2 Ghz;
- 1 Gigabyte di RAM;
- 1 disco EIDE (120 Gigabyte per il "master", 80 Gigabyte per gli altri nodi);
- 2 interfaccie di rete

In Figura 1 viene mostrata l'architettura di comunicazione del cluster. La macchina è stato costruita avendo in mente di rispettare alcuni requisiti. Il sistema è stato strutturato su due parti logiche distinte.
La prima parte, formata dal nodo master, rappresenta il front-end del cluster e permette agli utenti di collegarsi al sistema. Per l'accesso è stato scelto di utilizzare il servizio `ssh` in modo da fornire una shell sicura. Sul front-end sono inoltre presenti tutti gli strumenti di sviluppo necessari per la realizzazione delle applicazioni da testare sul cluster. Il nodo master ha anche il compito di gestire la *home* di tutti gli utenti e di renderla disponibile a tutti i nodi interni.

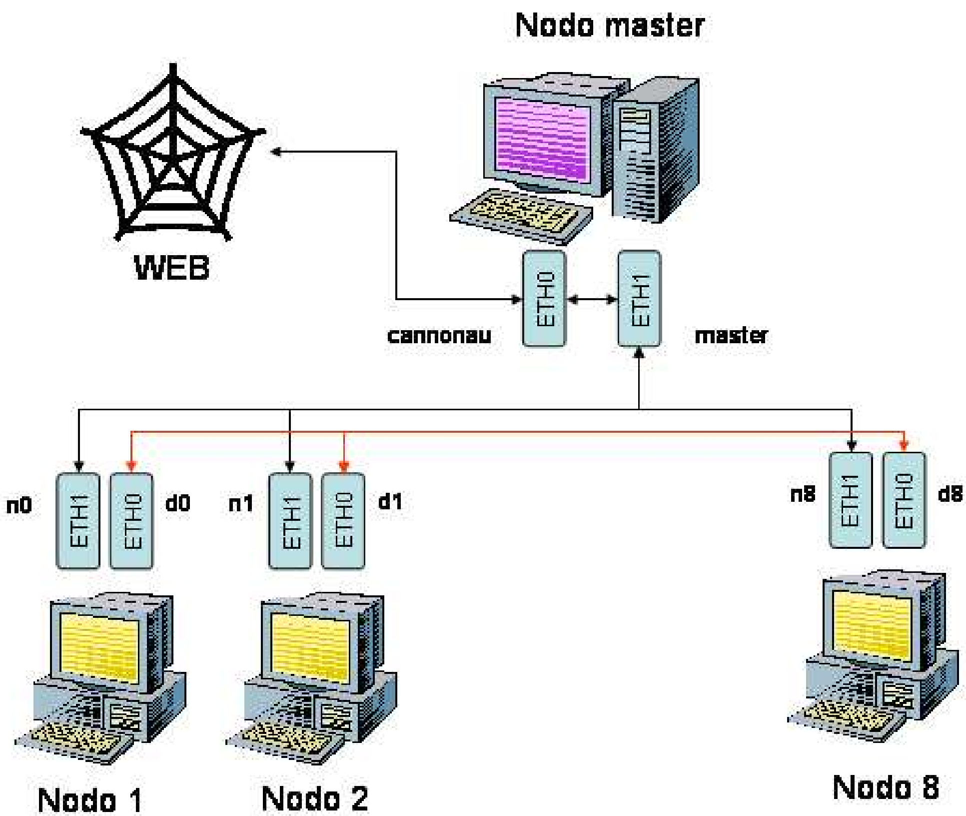

Figura 1: L'architettura logica del cluster

La seconda parte logica è formata dai nodi interni e mette a disposizione di ciascun utente le risorse di calcolo del cluster. Per sfruttare le risorse di ciascun nodo, ogni utente ha a disposizione sostanzialmente due possibili alternative. La prima consiste nel loggarsi sulla macchina che vogliamo utilizzare ed eseguire a mano le applicazioni che vogliamo testare. La seconda possibilità consiste nel sottomettere il nostro job al sistema di code (PBS) installato sul cluster. In quest'ultimo caso, sarà il sistema di code che deciderà per noi (in modo compatibile alla richiesta eseguita) dove e quando eseguire l'applicazione.

Ogni nodo del sistema di clustering presenta due interfaccie di rete.
Per quanto riguarda i nodi interni, la sottorete definita dall'interfaccia `eth0` è usata internamente dai programmi in esecuzione sul cluster e permette a questi ultimi lo scambio di dati nel modo più efficiente possibile grazie allo sfruttamento ottimale della banda disponibile. E' importante osservare che questa sottorete è accessibile esclusivamente dall'interno del cluster.
L'interfaccia `eth1`, invece, è utilizzata per connettere ogni nodo slave ad Internet. Questo è possibile impostando su questi ultimi, come gateway di default, l'indirizzo dell'interfaccia `eth1` del nodo master. Quest'ultimo nodo è collegato direttamente alla Rete tramite `eth0` e, grazie all'attivazione dei servizi di IP-masquerading e IP-forwarding, permette al traffico IP ricevuto dai nodi di essere inoltrato su Internet.
Il front-end del cluster, a differenza degli altri nodi, è collegato direttamente ad Internet tramite `eth0`. L'interfaccia `eth1` serve esclusivamente per far comunicare il front-end con i nodi interni e permette, come gia detto in precedenza, di condividere sia l'accesso a Internet sia eventuali altre risorse (ad esempio punti di mount NFS).

2 Modalità di accesso al cluster

Il cluster è accedibile dall'esterno attraverso il nodo front-end (cannonau.isti.cnr.it, 146.48.82.190). Per loggarsi nel sistema è necessario utilizzare il servizio *ssh*. Questo permette di mettere a disposizione di ogni utente autorizzato una shell criptata. Per esempio, per collegarsi al cluster come utente "utente" eseguire

```
ssh utente@cannonau.isti.cnr.it
```

e, quando richiesto, digitare la password corretta. Ricordiamo che per richiedere un account valido è necessario contattare gli amministratori del sistema (vedi sezione 4).

Una volta loggati nel sistema, è possibile accedere ai vari nodi del cluster attraverso il servizio *rsh*. Ogni nodo interno, come è stato descritto nella sezione 1, possiede due interfacce di rete che vengono utilizzate per creare due reti distinte.

La prima, pubblica e visibile anche dal nodo master, permette di far comunicare quest'ultimo con tutti gli altri nodi. In questo caso, ogni nodo ha un nome del tipo `nX` dove `X` è il numero di nodo considerato, antecedendo lo 0 nel caso `X` sia minore di 10.

La seconda rete è accessibile ai soli nodi interni e deve essere usata per far comunicare applicazioni distribuite in esecuzione sul cluster. I nomi utilizzati all'interno di questa rete seguono il formato `dX` dove il parametro `X` ha lo stesso significato del caso precedente.

Alla luce di quanto è stato detto, è possibile accedere ad un nodo interno, a partire dal master, eseguendo `rsh nX` dove X è il nodo da accedere. Viceversa, da ogni nodo interno è possibile accedere a ciscun altro nodo eseguendo sia `rsh dX` sia `rsh nX` dove `X` è la macchina destinazione.

La *home* di ogni utente è esportata in NFS dal nodo master su tutti i nodi. Questo permette di semplificare la gestione dei dati perchè i file locali sono acceduti in modo centralizzato indipendentemente dal nodo su cui un utente è loggato. Su ciascun nodo è inoltre disponibile una directory locale (in `/extra` e corrispondente al nome dell'utente) che può essere utilizzata per memorizzare dati critici che necessitano, per motivi di efficienza, di essere acceduti direttamente dall'hard-disk.

3 Operazioni usuali svolte dall'utente del cluster

3.1 Utilizzo della "broadcast shell"

Il comando non convenzionale *brsh* è l'abbreviazione di "broadcast shell" e serve per eseguire un comando simultaneamente su tutti i nodi del cluster. Si tratta di uno script di shell che viene usato nel seguente modo: `brsh <comando_da_eseguire>`
Ad esempio, per sapere quali utenti sono loggati sul cluster e cosa stanno facendo, basta eseguire "brsh w". L'output generato sarà qualcosa tipo il seguente:

```
*** n01 ***
  4:14pm  up 25 days, 1:39, 2 users, load average: 1.00, 1.00, 1.43
USER     TTY      FROM             LOGIN@   IDLE    JCPU  PCPU  WHAT
ecd      pts/0    master          12:23pm  1:47m  23:41   0.55s tail -f test.txt
ecd      pts/1    master           1:14pm  5:44    0.03s  0.03s -bash
*** n02 ***
  4:14pm  up 25 days, 1:39, 0 users, load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE    JCPU  PCPU  WHAT
*** n03 ***
  4:14pm  up 25 days, 1:39, 0 users, load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE    JCPU  PCPU  WHAT
*** n04 ***
  4:14pm  up 25 days, 1:39, 0 users, load average: 0.00, 0.00, 0.00
```

```
USER    TTY     FROM             LOGIN@   IDLE    JCPU  PCPU  WHAT
*** n05 ***
  4:11pm  up4:51,  0 users, load average: 0.00, 0.00, 0.00
USER    TTY     FROM             LOGIN@   IDLE    JCPU  PCPU  WHAT
*** n06 ***
  4:14pm  up 23 days, 6:10, 1 users, load average: 0.00, 0.00, 0.00
USER    TTY     FROM             LOGIN@   IDLE    JCPU  PCPU  WHAT
root    tty/1   -               26Feb03 23days   0.02s 0.02s  -bash
*** n07 ***
  4:13pm  up 23 days, 6:04, 0 users, load average: 0.00, 0.00, 0.00
USER    TTY     FROM             LOGIN@   IDLE    JCPU  PCPU  WHAT
*** n08 ***
  4:14pm  up 25 days, 2:00, 0 users, load average: 0.00, 0.00, 0.00
USER    TTY     FROM             LOGIN@   IDLE    JCPU  PCPU  WHAT
```

Lo script, se eseguito in questo modo, accede in sequenza ai vari nodi ed esegue il comando specificato. Così facendo, un comando da eseguire sul nodo i non può iniziare fino a che il comando sul nodo i-1 non è terminato (con i =2..8). Per parallelizzare le operazioni, basta eseguire lo script passandogli un comando da eseguire in background. Ad esempio

```
brsh comando_da_eseguire&
```

3.2 Utilizzo del compilatore GCC

Il software GCC è un compilatore gratuito, rilasciato sotto licensa OpenSource, che attualmente rappresenta lo standard defacto per lo sviluppo di applicazioni in ambiente Linux (e Unix-like in generale). Il tool è in grado di compilare efficentemente programmi scritti in vari linguaggi di programmazione: oltre ai classici C/C++ e Java, sono supportati anche altri linguaggi meno "popolari" quali ADA, Fortran ed Objective-C.

Generalmente, il compilatore GCC non viene utilizzato direttamente da shell bensì mediante GNU Make, uno strumento che serve a gestire progetti software di una certa complessità. Il file di progetto utilizzato dal comando *make* viene chiamato Makefile e consiste in semplice file di testo contenente direttive e comandi da eseguire. Un possibile esempio per un semplice progetto software potrebbe essere il seguente:

```
# Un semplice Makefile per un programma scritto in C

# Definisco la variabile compilatore
CC=gcc
```

```
# Definisco i flag da passare al compilatore
CFLAGS=-c -Wall -O

# Definisco le librerie da utilizzare. Aggiungere le opzioni -L -l
# per ciascuna libreria utilizzata
LIBS=-L/path/to/mylib -lmylib

# Definisco i file oggetto da creare a partire dai sorgenti
OBJS=file1.o file2.0 file3.o

# Le direttive sono della forma:
# <obiettivo>:<TAB><dipendenze_obiettivo>
# <TAB><comandi_da_eseguire>
all: $(OBJS)
        $(CC) -o nome_eseguibile $(LIBS) $(OBJS)

file1.o file1.c file1.h
        $(CC) $(CFLAGS) file1.c

file2.o file2.c file2.h
        $(CC) $(CFLAGS) file2.c

file3.o file3.c file3.h
        $(CC) $(CFLAGS) file3.c
```

Nella parte iniziale del precedente Makefile vengono definite alcune viariabili che saranno utilizzate successivamente nell'esecuzione dei comandi. Osserviamo che il loro utilizzo (la definizione) non è strettamente necessario però è vantaggioso perchè permette di scrivere un Makefile "più pulito" e più semplice da modificare. Ad esempio, nel caso volessimo utilizzare un diverso compilatore basterebbe semplicemente cambiare il valore della variabile *CC*.
Le direttive vengono interpretate a partire dalla prima trovata all'interno del file. Nell'esempio precedente, *make* dapprima prova a risolvere *all*. Analizzando la direttiva, si accorge che vi sono varie dipendenze ($(OBJS)) che devono essere risolte. In sequenza, quindi, espande la variabile OBJS e interpreta allo stesso modo le singole direttive trovate nell'ordine (*file1.o, file2.o file3.o*). Nel caso una o più direttive dipendano da file sorgenti modificati rispetto all'ultima compilazione, viene di nuovo eseguita la compilazione dei sorgenti e creato un nuovo file oggetto(.o). Alla fine, dopo aver controllato tutte le direttive, viene rigenerato, se necessario, l'eseguibile del progetto. Di seguito elenchiamo i flag di compilazione più comunemente utilizzati:

-o `<nome_eseguibile>` Viene utilizzato per cambiare il nome del file eseguibile creato dal compilatore.

-c Indica al compilatore di creare un file oggetto, senza effettuare il linking alle librerie utilizzate.

-l `<libreria>` Indica al compilatore di linkare la libreria dinamica `<libreria>`. Di solito, viene utilizzato insieme al flag *-L*.

-L `<path_libreria>` Indica la directory nella quale cercare una particolare libreria dinamica.

-I `<path_include>` Indica la directory nella quale cercare dei particolarei header file.

-Wall Abilita il compilatore a segnalare tutti i possibili warning verificatesi durante la conpilazione.

-O Abilita il massimo livello di ottimizzazione del codice generato.

-g Abilita il supporto di debug. In tal modo è possibile utilizzare *gdb* per fare il debugging del programma.

Per avere maggiori dettagli sui flag descritti e su tutte le possibili opzioni disponibili, potete consultare direttamente la pagina di manuale del GCC.

3.3 Gestione dei job in PBS

Il software PBS (Portable Batch System) è un sistema di gestione batch dei job conforme allo standard POSIX 1003.2d Bath Environment Standard. Il suo compito è quello di accettare dei lavori in ingresso (sottoposti ad alcuni attributi di controllo) e successivamente, quando le condizioni di esecuzione sono soddisfatte, di eseguirli facendosi carico di ritornare l'output generato dai job agli utenti che li avevano sottomessi.
I job devono essere sottomessi ESCLUSIVAMENTE dal nodo master utilizzando il comando `qsub`. Questo prende in input uno script di shell in cui, oltre a specificare il processo che vogliamo eseguire, possiamo indicare alcuni parametri utilizzati dal sistema di code per imporre alcuni vincoli sull'applicazione che deve essere eseguita. Un possibile esempio di script è

```
#!/bin/sh

# Questo e' un esempio di script da sottomettere a PBS
# Gli script possono contenere commenti (righe che iniziano con "#"),
# direttive per PBS (righe che iniziano con "#PBS") e comadi in chiaro.
```

```
# Supponiamo di voler eseguire questo job alle ore 12:30 di oggi.
# PBS -a 1230

# Assegno il nome "Nome_job" a questo job anziche' utilizzare quello
# di default impostato da PBS
# PBS -N Nome_job

# Il processo da eseguire
processo_da_eseguire
```

Le direttive da utilizzare nei file di script ccorrispondono alle opzioni da riga di comando passabili al comando `qsub`. Per un elenco completo dei possibili parametri, potete consultare direttamente la pagina di manuale di questo comando. Per agire direttamente sui job sottomessi a PBS vi sono svariati comandi (utilizzabili esclusivamente dal nodo master). Di seguito elenchiamo i più importanti:

qstat Permette di ottenere informazioni statistiche sui job sottomessi, sullo stato delle code e dei server batch.

qalter Permette di modificare gli attributi associati ad un particolare job.

qdel Permette di cancellare un job sottomesso in precedenza.

qmsg Serve ad inviare un messaggio (una stringa) ad un particolare job. Solitamente, la stringa rappresenta una nota informativa e viene appesa su uno dei file di output del processo.

qmove Muove un job dalla coda dove attualmente si trova ad un'altra specificata.

qselect Visualizza una lista di identificativi di job che corrispondono al criterio di ricerca specificato.

qsignal Invia un segnale ad uno specifico job.

3.4 Utilizzo di LAM

Per compilare applicazioni MPI è necessario utilizzare il comando `mpicc` nel caso i sorgenti siano in C, `mpiCC` nel caso i sorgenti siano in C++ oppure `mpif77` nel caso i sorgenti siano in Fortran. I comandi suddetti sono dei wrapper sui tool di compilazione classici (ad esempio `gcc`) che impostano automaticamente tutte le librerie necessarie per compilare correttamente un'applicazione. Nel caso un programma necessiti di altre librerie esterne, è possibile utilizzare i flag standard del gcc. Ad esempio:

```
mpicc -lmia_libreria -L/path/to/mia_libreria prova.c
```

Ogni utente che vuole eseguire un'applicazione MPI deve, per prima cosa, impostare un opportuno ambiente run-time su cui eseguire i processi. In particolare, è necessario indicare a LAM quali macchine si intendono utilizzare. Se non diversamente specificato, la configurazione standard prevede l'utilizzo di tutti i nodi del cluster. Per utilizzare una macchina virtuale personalizzata, l'utente deve creare un file (ad esempio *lamhosts*) su cui inserire il nome delle macchine desiderate:

```
# IMPORTANTE - Per sfruttare la sottorete privata di
comunicazione tra i nodi interni del cluster, non e'
possibile utilizzare come nodo computazionale il
master

# In questo esempio utilizzo solo il nodo 3 e 5. LAM
# identifichera' il nodo 3 con n0 e il nodo 5 con n1
d03
d05
```

Adesso attiviamo l'ambiente LAM con le impostazioni precedenti. Lo possiamo fare in due modi diversi. Il primo quello di loggarsi su uno dei nodi specificati nel file mediante il comando `rsh` ed eseguire

```
lamboot -v lamhosts
```

Il secondo metodo consiste nell'eseguire il comando "lamboot" sfruttando direttamente il comando rsh, ad esempio

```
rsh n03 lamboot -v lamhosts
```

Come già detto in precedenza, in entrambi i casi il comando deve essere eseguito da uno dei nodi specificati nel file di configurazione. Questo è necessario affinchè sia utilizzata la sottorete privata per le comunicazioni tra i processi in esecuzione sui nodi della macchina virtuale. A questo punto, per eseguire la nostra applicazione MPI utilizziano il comando mpirun. Ad esempio

```
mpirun -v -np 4 mio_eseguibile
```

mette in esecuzione sulla macchina virtuale specificata 4 copie del programma `mio_eseguibile`. Se la nostra applicazione è formata da più eseguibili è necessario descriverla attraverso un cosiddetto "application schema". Ad esempio, supponiamo di voler eseguire un sistema client-server contenente un processo master ed uno slave. Lo schema rappresentante la descrizione di questa particolare applicazione potrebbe essere il seguente file (di nome `app_schema`):

```
# Il master viene eseguito sul primo nodo della macchina virtuale
n0 programma_master

# Due copie dello slave sono eseguite sul secondo e terzo nodo
#della macchina virtuale
n1-2 programma_slave
```

Il suddetto file può essere direttamente passato come parametro al comando `mpirun`:

```
mpirun -v app_schema
```

Lo stato dei processi MPI in esecuzione su LAM può essere monitorato attraverso l'utilizzo della utility `mpitask`. Se non viene specificato un task specifico, il comando visualizzerà informazioni statistiche sullo stato di tutti i processi in esecuzione su tutti i nodi del cluster virtuale. Per avere, invece, informazioni statistiche sullo stato dei buffer utilizzati per i messaggi è necessario usare il comando `mpimsg`.

Gli ultimi due comandi utili che menzioniamo sono `lamclean` e `lamhalt`. Il primo viene utilizzato per terminare tutti i processi dell'utente sul cluster(escluso il sistema run-time di LAM), in modo da deallocare correttamente tutte le risorse utilizzate. Il comando è utile, ad esempio, per terminare in modo pulito un'applicazione contenente alcuni bug.
Il comando `lamhalt` viene invece utilizzato per terminare il sistema run-time di LAM. Questa operazione deve essere eseguita quando non è più necessario eseguire alcun programma MPI.

4 Contatti

Per la richiesta di nuovi account e chiarimenti sulla configurazione ed utilizzo del cluster, potete contattare gli amministratori del sistema:

- Giancarlo Bartoli (giancarlo.bartoli@isti.cnr.it)
- Tiziano Fagni (tiziano.fagni@isti.cnr.it)
- Salvatore Orlando (orlando@dsi.unive.it)
- Paolo Palmerini (paolo.palmerini@isti.cnr.it)
- Raffaele Perego (raffaele.perego@isti.cnr.it)
- Fabrizio Silvestri (fabrizio.silvestri@isti.cnr.it)