Linux per la ricerca

Miniguida all'uso del Cluster di Calcolo

Giovanna Neve, Silvia Sartorelli

Versione maggio 2011

Introduzione

Questo manuale è stato scritto dai tecnici informatici del Dipartimento di Scienze Statistiche dell'Università di Padova in seguito alle sempre più numerose richieste di utilizzo di *server* dedicati al calcolo e alla ricerca. I destinatari sono quindi ricercatori, dottorandi o collaboratori alla ricerca con limitata esperienza nell'uso di macchine (remote) con sistema operativo *Unix-like*, come ad esempio GNU/Linux.

Rispetto alla versione precedente, sono stati aggiornati i riferimenti espliciti alla macchina ma soprattutto il manuale è stato arricchito di un'appendice, scritta dal prof. Matteo Grigoletto, a cui vanno i nostri ringraziamenti, in cui si spiega come utilizzare il cluster di calcolo sfruttando la possibiltà di compiere calcolo parallelo.

Per prima cosa, per accedere ad una risorsa di questo tipo, l'utente deve conoscerne esattamente il nome ed avere un account su tale macchina. Questo elaborato fornirà quindi strumenti per:

1. trasferire i propri file dall'abituale ambiente di lavoro alla macchina remota e viceversa (vedi Paragrafo 1.1);
2. accedere alla macchina remota (vedi Paragrafo 1.2);
3. eseguire semplici operazioni in ambiente Linux (vedi Capitolo 2);
4. modificare file di testo in ambiente Linux (vedi Capitolo 3).

Si potranno quindi sfruttare appieno le potenzialità della macchina di calcolo eseguendo i propri programmi in modalità *batch*.

Negli ultimi capitoli e nell'appendice si farà espressamente riferimento all'utilizzo del cluster di calcolo del nostro Dipartimento.

Capitolo 1

Accedere e trasferire file a/da macchine remote Linux

1.1 Trasferire file

Per trasferire file tra macchine remote è innanzitutto necessario che entrambi i computer siano in grado di utilizzare lo stesso protocollo di comunicazione sicura. Vediamo ora i software più utilizzati, a seconda del sistema operativo.

1.1.1 da Windows

WinSCP è un programma per il trasferimento sicuro di file (SFTP - Secure File Transfer Protocol). La sua funzione principale è quella di copiare in modo sicuro file tra un computer locale e uno remoto. Si trova sul DVD di Facoltà (reperibile in ASID) oppure è possibile scaricarlo dal sito:
http://winscp.net/eng/docs/lang:it

Una volta installato e lanciato, è necessario inserire nell'interfaccia per la connessione il nome del server, il nome utente e la password.

Effettuata l'autenticazione, si apre la finestra del programma che presenta due riquadri: a sinistra viene visualizzato il computer locale e a destra il computer remoto. Nella barra in basso sono riportate tutte le funzioni principali: Rinomina, Modifica, Copia, Sposta, Crea Cartella, Elimina, Proprietà, Esci. Ci si può muovere tra le varie directory in entrambi gli ambienti, crearne di nuove, etc.

1.1.2 da Linux

Se si usa una macchina Linux, si può utilizzare il comando **scp** da una finestra di terminale:

```
mioUser@miaMacchina:~$ scp file userRemoto@macchinaRemota:
```

quindi verrà chiesta la password sulla macchina remota. In questo modo il file viene trasferito nella home dell'utente nella macchina remota. Eventualmente è possibile specificare un'altra posizione (vedi Capitolo 2).

1.1.3 da MAC

I comandi che permettono agli utenti MAC di comunicare con macchine remote sono gli stessi disponibili per gli utenti Linux: il comando **scp** da una finestra di terminale è utilizzato per il trasferimento di file:

```
miaMacchina:~mioUser$ scp file userRemoto@macchinaRemota:
```

Anche in questo caso è possibile specificare una posizione di destinazione diversa dalla home inserendone il percorso.

1.2 Accedere ad una macchina remota

Si è visto come trasferire file su una macchina remota. Per operare su di essa, è necessario disporre di programmi che presentino una finestra di terminale del computer remoto. Vediamo ora quali sono i più utilizzati, a seconda del sistema operativo.

1.2.1 da Windows

PuTTY è un programma per gestire connessioni SSH. Si trova sul DVD di Facoltà oppure è possibile scaricarlo dal sito:
http://www.chiark.greenend.org.uk/~sgtatham/putty/

Una volta installato e avviato, richiede vari parametri; in particolare va compilato con:

```
Host name (or IP address): user@macchinaRemota
Port: 22
Connection type: SSH
```

Putty permette la comunicazione anche tramite protocolli diversi da SSH, ma essendo di tipo non sicuro, se ne sconsiglia l'uso.
Successivamente vengono richieste le credenziali di accesso (nome utente e password) e, se l'autenticazione va a buon fine, si presenterà il prompt Linux

```
user@macchinaRemota:~$
```

Il comando per terminare la connessione è:

```
user@macchinaRemota:~$ exit
```

1.2.2 da Linux

Per accedere alla shell della macchina remota Linux, il comando che va digitato da una finestra di terminale è **ssh**:

```
mioUser@miaMacchina:~$ ssh user@macchinaRemota
```

Anche qui andranno inserite le credenziali sulla macchina remota. Il comando per terminare la connessione è

```
user@macchinaRemota:~$ exit
```

1.2.3 da MAC

Il comando che permette agli utenti MAC di comunicare con macchine remote sono gli stessi disponibili per gli utenti Linux; per la connessione si utilizza **ssh** da una finestra di terminale:

```
miaMacchina:~mioUser$ ssh user@macchinaRemota
```

e

```
user@macchinaRemota:~$ exit
```

per uscire.

Capitolo 2

Destreggiarsi in ambiente Linux

In ambienti Linux i comandi hanno la forma

```
user@macchina:~$ comando -eventualiOpzioni eventualiArgomenti
```

Qui di seguito trovate un elenco dei principali comandi con le opzioni più utilizzate. Per avere una descrizione dettagliata di ciascun comando, si faccia riferimento al manuale richiamabile in qualsiasi momento col comando

```
user@macchina:~$ man comando
```

Cominciamo con la visualizzazione di file nelle directory:

`pwd`	mostra il path della directory corrente
`cd directory`	cambia directory. Es `cd lavori/2009` posiziona nella sottodirectory 2009 contenuta nella directory lavori
`ls`	elenca il contenuto della directory corrente
`ls -l`	come prima, mostrando alcuni dettagli sui file (es. tipo di file, permessi, proprietario, dimensioni, data ultima modifica)
`ls -la`	come prima, ma includendo nella lista anche i file nascosti
`ls -l file`	mostra dettagli sul file specificato

L'output del comando ls -l visualizza una serie di informazioni molto importanti. Ad esempio:

```
-rwxr-xr-- 2 user staff  123456 2009-07-06 10:51 Esempio.txt
```

Nella prima colonna, il primo simbolo ("-" in questo caso) indica un attributo specifico del file ("-" se file, "d" se directory, ecc.). Seguono tre terne di simboli indicanti ciascuna i permessi relativi al proprietario (u, user), al gruppo (g, group), a tutti gli altri (a, all). In ciascuna terna (`rwx`), con r si indica il permesso di lettura del file per visualizzarne il contenuto (o di visualizzare il contenuto della directory);

con w il permesso di scrittura, che consente di sovrascrivere o aggiungere dati a un file (o di crere o di eliminare di file all'interno della directory); con x il permesso di esecuzione che consente di eseguire un file, nel caso si tratti di un file eseguibile (o di accedere alla directory, per esempio con il comando `cd`, anche nel caso non si possa visualizzare il contenuto della directory). Nell'esempio sopra riportato, si ha un file chiamato Esempio.txt, in cui l'utente (user) ha il permesso di lettura, scrittura ed esecuzione, il gruppo (staff) solo lettura ed esecuzione, mentre tutti gli altri solo lettura; si tratta di un file di 123456 byte, salvato l'ultima volta il 6 luglio 2009 alle 10.51.

`chown newpropr file`	cambia il proprietario del file. Con l'opzione `-R` modifica il proprietario di file e directory ricorsivamente
`chmod oper perm file`	modifica i permessi sul file. Operatori: u, g, a: user, group, all. Permessi: r, w, x: lettura, scrittura, esecuzione, operatori + (aggiungi), - (elimina) o = (imposta). Esempio: chmod u+wx prova.txt: assegna all'utente permesso di modifica ed esecuzione. Esempio: chmod a-rwx: elimina tutti i permessi altri utenti. Esempio: chmod g=rw: imposta i permessi di lettura e modifica al gruppo.

Comandi per la creazione, la copia e la rimozione di file e directory:

`mkdir directory`	crea directory. Es.: `mkdir /home/utente/miaDirectory`. Se non viene specificato il path, la directory viene creata nella posizione corrente
`mv fileSorg fileDest`	sposta il file. È possibile scrivere il path completo sui nomi dei file sorgente e destinazione. Es: mv /home/gianni/example.txt /home/carlo/ Se sorgente e destinazione sono sulla stessa directory, si ha una ridenominazione. Es.: mv pippo.txt pluto.txt rinomina pippo.txt in pluto.txt
`cp fileSorg fileDest`	crea una copia. Come prima, è possibile scrivere il path completo sui nome dei file sorgente e destinazione

`rm file`
`rm *`
 	
`rm -r directory`
	elimina il file (o la directory)
	elimina il contenuto della directory corrente. Attenzione! Meglio specificare sempre il path. Es.: `rm /home/utente/*` elimina tutti i file nella cartella utente
	rimuove la directory e le sottodirectory

Comandi per la comunicazione tra macchine remote:

`scp file1 user@host:file2`	copia file tra hosts nella rete tramite ssh. Es.: `scp miofile utente@server`. (Vedi anche Sezione 1.1.2)
`ssh user@server`	apre un client SSH (Vedi anche Sezione 1.2.2)

`screen`
 	
`screen --comando`
 	
`screen -r`
 	
`screen -ls`
	emula un terminale; permette di sganciare un processo dalla sessione, per poi riagganciarlo in un secondo momento
	esegue il comando in una shell separata. Con CTRL+a e poi d ci si sgancia dalla sessione corrente lasciando attivi i processi
	recupera una sessione precedentemente sganciata
	visualizza le sessioni attive

Comandi per la visualizzazione del contenuto di file:

`cat file`	visualizza il contenuto del file
`more file`	visualizza il contenuto del file ma suddividendolo in pagine
`less file`	come prima, ma permette di scorrere il documento anche indietro (q per uscire)

E ora un po' di comandi per la gestione dei processi:

`ps`
`ps ux`
 	
`ps faux`
	elenca i processi attivi
	mostra i processi dell'utente. Vengono fornite informazioni fra cui il process ID (o PID)
	mostra i processi del sistema

`kill -9 PID`	termina il processo identificato dal numero PID
`top`	mostra i task in tempo reale (q per terminare)

Quando viene lanciata l'esecuzione di un programma, è utile poter eseguire altre operazioni, sebbene non sia terminata o sospesa la prima esecuzione (questo non è concesso se non viene esplicitamente dichiarato). Vediamo come fare:

`comando &`	esegue il comando in background
`^ z`	sospende (mette in "pausa") il processo corrente
`^ c`	interrompe il processo corrente
`fg`	riporta il processo in primo piano
`bg`	esegue in background un processo precedentemente interrotto
`jobs`	mostra i processi attivi in background

Per gestire la priorità con cui vengono eseguiti i programmi si utilizzano i seguenti comandi:

`nice prior comando`	esegue un programma con priorità modificata. La priorità va da -20 (più prioritario) a +19 (meno prioritario)
`nice -n N comando`	aumenta (o diminuisce) di N la priorità
`renice prior PID`	imposta la priorità al processo PID. Vedi comando nice per la gestione delle priorità
`nohup comando`	Evita l'interruzione del processo quando per problemi di rete o normale operatività ci si sconnette dalla sessione remota. Tutti gli output che verrebbero stampati a schermo vengono scritti sul file nohup.out

Capitolo 3

Modificare file di testo

Illustriamo brevemente l'utilizzo dei due editor di testo più diffusi, lasciando al lettore l'eventuale approfondimento sulla documentazione ufficiale.

3.1 VIM

Per aprire l'editor VIM, digitare da riga di comando

```
user@macchina:~$ vim nomeFile
```

Si apre una finestra vuota, se il file è appena stato creato.

Esistono tre modalità (inserimento, sostituzione, comando) che vengono segnalate sull'ultima riga della schermata, a sinistra: "INSERISCI", "SOSTITUISCI" oppure niente, per la modalità comando.

Per cominciare a digitare del testo è necessario mettersi in modalità "inserimento", premendo la lettera i.

Comandi per spostarsi all'interno di un testo. I seguenti comandi vanno digitati in modalità comando.

`0`	sposta il cursore all'inizio della riga
`$`	sposta il cursore alla fine della riga
`G`	sposta il cursore alla fine del documento
`:n`	posiziona il cursore alla riga n-esima

Comandi per la cancellazione e la correzione:

`dw`
`dd`
`3dd`
	cancella la parola seguente
	cancella la riga corrente
	cancella le 3 righe seguenti. NB: il comando d può essere utilizzato con funzione "taglia"

`:u`	annulla la cancellazione precedente

Comandi per la ricerca, copia e sostituzione:

`/parola`	ricerca la parola all'interno del testo. Premere n per proseguire la ricerca

`yw`
`Y`
`y$`
	copia la parola corrente
	copia la riga corrente
	copia dal cursore alla fine della riga

`p`	incolla
`:%s/prova/PROVA/gc`	sostituisce la stringa prova in PROVA in tutto il file, anche più volte nella stessa riga, chiedendo conferma ogni volta

Comandi per la manipolazione dei file:

`:w`	salva il file
`:q`	esce dall'editor
`:wq`	salva ed esce
`:shell`	lancia una shell all'interno dell'editor. CTRL+d per uscire

Per ottenere lettere accentate (à, è, ecc) bisogna digitare \`a, \`e, ecc.

$, &, %, #, _, {, } sono caratteri speciali che devono essere preceduti da backslash:

`\$, \&, \%, \#, \_, \{, \}`

Invece, per ottenere ~, ^, \, bisogna digitare

```
\~{}, \^{}, $\backslash$
```

3.2 emacs

Per aprire l'editor emacs, digitare da riga di comando

```
user@macchina:~$ emacs nomeFile
```

Ecco un elenco dei comandi più utilizzati. Per un approfondimento, consultare la documentazione ufficiale.

`CTRL+d`	cancella dalla posizione del cursore alla fine della riga
`ALT d`	cancella una parola
`CTRL+y`	incolla il testo eliminato
`ALT q`	riformatta il paragrafo
`CTRL+s stringa`	ricerca la stringa
`CTRL+xs`	salva il file
`CTRL+xc`	uscita

Capitolo 4

Utilizzo del cluster di calcolo

4.1 Che cos'è un cluster di calcolo?

Il cluster di calcolo è ospitato su una macchina il cui nome è *Hactar*, composta da un 1 server e 7 lame (blade); tale macchina, gestita dal Dipartimento di Scienze Statistiche, è riservata all'elaborazione dati e al calcolo. È dotata di 64 CPU a 64bit, a 2.6GHz e con 16 GB di RAM per blade.

4.2 Che cosa serve per utilizzarlo?

Per accedere alla risorsa è necessario:

1. avere un account sulla macchina, che si richiede via helpdesk [1]. Al ricevimento della conferma della creazione dell'account, sarà possibile autenticarsi utilizzando le stesse credenziali della posta elettronica del Dipartimento;
2. avere uno strumento per il trasferimento dei propri file dall'abituale ambiente di lavoro ad Hactar e viceversa (vedi Paragrafo 1.1);
3. avere uno strumento per accedere ad Hactar (vedi Paragrafo 1.2);
4. sapersi destreggiare in ambiente Linux (vedi Capitolo 2);
5. poter modificare file di testo (vedi Capitolo 3);
6. sapere usare R in modalità *batch* e impostare la priorità dei propri processi (vedi Capitolo 5);
7. conoscere alcune nozioni sul calcolo parallelo con R per sfruttarne al meglio le potenzialità (vedi Appendice A)

[1] per gli studenti: la richiesta, motivata, deve venire inoltrata dal relatore, sentito il responsabile ASID. Nell'oggetto della mail scrivere "richiesta account cluster" e nel corpo specificare nome, cognome e posizione all'interno dell'Ateneo.

Capitolo 5

Come usare R su Hactar

Per avviare R su Hactar basta dare il comando "R" dal prompt Linux:

```
user@hactar:~$ R
```

e comparirà il conosciuto prompt per l'uso di R a linea di comando:

```
R version 2.9.0 (2009-04-17)
Copyright (C) 2009 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
(...)
[Previously saved workspace restored]
>
```

e per uscire il solito:

```
>q()
```

In particolare noi siamo interessati ad eseguire script già pronti; per mandare in esecuzione i comandi possiamo scegliere tra due sintassi possibili:

```
R[options][<infile][>outfile]
```

oppure

```
R CMD comandi argomenti
```

Quest'ultima sintassi va usata in modalità *batch* quindi, supponendo di avere un programma "esempio1R.txt" precedentemente trasferito su Hactar, la sintassi sarà:

```
user@hactar:~$ R CMD BATCH esempio1R.txt
```

Al termine del programma si troverà nella home un file "esempio1R.txt.Rout" e, se lo script lo prevede, vi troveremo anche i grafici o qualsiasi altro file prodotto.

Con la prima sintassi invece, supponendo di avere lo stesso script "esempio1R.txt" e di voler salvare l'output nel file di testo "out.txt" scriveremo;

```
user@hactar:~$ R --save <esempio1R.txt >out.txt
```

Analogamente a quanto visto in precedenza, se lo script prevede la produzione di grafici, etc, troveremo tali prodotti nella directory di lavoro. Hactar però non ha un display grafico, quindi è necessario spostare i file via scp (winscp) sulla usuale macchina windows/ubuntu e visualizzarli da lì.

L'opzione `--save` serve per salvare i dataset al termine della sessione; è necessario specificare se si desidera farlo oppure no (`--no-save`). Se l'esecuzione dei comandi lanciati durerà giorni, è ovvio che si desidererà chiudere la connessione dopo averli lanciati e tornare dopo qualche giorno per controllare come procede ed evenutalmente ritirare i risultati. In questo caso occorre usare il comando `screen`:

```
screen -S <nome_sessione>

#esegui r

CTRL+A e poi D #per sospendere la sessione

screen -list #per vedere le sessioni attive

screen -r <nome_sessione> #per ricollegarti alla sessione

CTRL+D # per terminare la sessione alla fine del lavoro
```

Per approfondire la sintassi e conoscere gli altri parametri possibili rimandiamo all'help in linea oppure all'"Appendice B: Invoking R" di "An Introduction to R" di Venables e Smith[1].

Per controllare il contenuto della directory, cancellare e/o modificare file direttamente su Hactar, si vedano i comandi nel Capitolo 2. Se si invia un programma su Hactar è necessario:

- farlo girare in background, completando il comando con un `&`
- per permettere un utilizzo proficuo per tutti gli utenti, modificare la priorità del processo avviato, abbassandola con il comando `renice 19 PID`[2] (si veda il Capitolo 2).

[1] vedi http://cran.r-project.org/doc/manuals/R-intro.pdf

[2] Process ID, per conoscerlo il comando è ps ux

Appendice A

Using the new cluster for parallel computing with R

This document gives a short description about the use of the new cluster available in the Department of Statistical Sciences, University of Padua. The procedure described below is not the only one available: the objective is just to give some hints to get started in parallel computing.

The cluster can be accessed, with an SSH client, at the following address

`cluster.stat.unipd.it`

or, equivalently

`hactar.stat.unipd.it`

The username and password are the same used for accessing email at the department server (`http://webmail.stat.unipd.it`).

For Windows users, a widespread SSH client is PuTTY, which can be downloaded freely.

After accessing `hactar.stat.unipd.it` and starting `R` (simply type "R" at the `$` Linux prompt), the commands below may be given.

First, we need to load some `R` libraries and define some variables:

```
require(snow); require(foreach); require(doSNOW);
version = paste("R-",R.Version()$major,".",R.Version()$minor,sep="")
rscriptdir = paste("/opt/",version,"/lib64/R/bin/Rscript",sep="")
snowlibdir = paste("/opt/",version,"/lib64/R/library",sep="")
```

In the following, 7 is the maximum number of slaves that can be used (they are called `blade1`, ..., `blade7`):

```
for (i in 1: 7) assign(paste("blade",i,sep=""),list(host = paste("blade",i,sep=""),
                        rscript = rscriptdir, snowlib = snowlibdir))
hactar = list(host = "hactar", rscript = rscriptdir, snowlib = snowlibdir)
host_list = list(blade1, blade2, blade3, blade4, blade5, blade6, blade7, hactar)
```

If we would like to user several (e.g., 4) CPU's for each slave, we may issue the following command:

```
host_list = rep(host_list,4)
```

Please, notice that this command may well slow down processes launched by other users, when not enough free CPU's are available. Hence, this is generally considered bad practice.

Here, we create the `R` cluster:

```
cl = makeCluster(host_list, master="hactar", type="SOCK")
```

We are now in a position to use parallel computing! First, we must register `DoSNOW`:

```
registerDoSNOW(cl)
```

As an example, we will apply the bootstrap technique. We are going to estimate 10000 GLM models (Weston, 2010).

```
x = iris[which(iris[, 5] != "setosa"), c(1, 5)]
trials = 1E4

ptime = system.time({
 r = foreach(icount(trials), .combine = cbind) %dopar% {
 ind = sample(100, 100, replace = TRUE)
 result1 = glm(x[ind, 2] ~ x[ind, 1], family = binomial(logit))
 coefficients(result1)
}
})[3]
ptime
```

Here, `ptime` shows the computation time.

A perhaps less intuitive, but more efficient way to reach the same result is:

```
parallel_func = function(ind,data)
{
   result1 = glm(data[ind, 2] ~ data[ind, 1], family = binomial(logit))
   coefficients(result1)
}

A = matrix(sample(100, 100*trials, replace = TRUE), nrow=trials)
ptime1 = system.time({r = parApply(cl, A, 1, parallel_func, data=x)})[3]
ptime1
```

Probably, the higher computational efficiency is due to the fact that with the latter approach we do not use `cbind`, that can be very time-consuming. The function `parApply` is a parallel version of `apply`. In particular, the function `parallel_func` takes each row of the matrix `A` as first argument (here, this argument is called `ind`). Other arguments can also be specified (here, we use `data = x`).

Should we need to load a library in each slave, we may use:

```
clusterEvalQ(cl, library(library_name))
```

By changing `%dopar%` to `%do%` in the foreach instruction above, we make the same computations sequentially, in order to determine the performance improvement obtained with parallel computations:

```
stime = system.time({
 r = foreach(icount(trials), .combine = cbind) %do% {
 ind = sample(100, 100, replace = TRUE)
 result1 = glm(x[ind, 2] ~ x[ind, 1], family = binomial(logit))
 coefficients(result1)
}
})[3]
stime
```

This shows the sequential computation time. We now shut down the cluster before exiting `R`:

```
stopCluster(cl)
```

Further details on parallel computing in `R` can be found in Rossini and Tierney (2003).

When using the above commands, there is an important caveat. It should be reminded that results are transferred from the slaves to the master, in order to be combined. This is a time-consuming task, and the time is directly proportional to the size of the results. In the above examples, we only transfer `coefficients(result1)`. It should be kept in mind that transferring large matrices would be highly inefficient, even to the point of making parallel computation useless!

If something goes wrong with the above commands and we get stuck somewhere, we may want to kill all the processes we have started in the slaves. Then, at the `$` Linux prompt (if necessary, we may start a new SSH session), we can issue the following command:

```
for i in {1..7}; do ssh -f "blade$i" killall -u $USER; done
```

To kill particular processes in the master computer (i.e. `hactar`), at the Linux prompt, look at the output of the command `top -u $USER`, press "q" and then use `kill PID` (where PID is the number of the process you want to kill). Critical processes you should kill are stuck `R` processes with high CPU usage (see column %CPU in the `top` output).

References

Weston, S. (2010). Getting started with doMC and foreach. Available at http://dssm.unipa.it/CRAN/web/packages/doMC/vignettes/gettingstartedMC.pdf

Rossini, A., Tierney, L., and Li, N. (2003). Simple parallel statistical computing in R. Technical Report Working Paper 193, UW Biostatistics Working Paper Series. http://www.bepress.com/uwbiostat/paper193

Indice