

<<<<<<< .mine ===== >>>>>>> .r1259

GRID@Trieste facile

Guida introduttiva all'uso della griglia

Riccardo Di Meo, Alessio Terpin, Antonio Messina

Copyright (©) 2005 **UNESCO**

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Sommario

1	Scopo di questo manuale	1
2	Concetti di base.....	2
2.1	Una breve panoramica	2
2.2	Componenti principali	2
2.2.1	Interazione con l'utente	2
2.2.2	Autenticazione e sicurezza	3
2.2.3	Meccanismo di funzionamento	3
2.2.4	Il sistema informativo interno	3
3	Preparazione dell'ambiente di lavoro (User Interface)	4
3.1	Requisiti minimi	4
3.2	Il Live CD di EGRID come User Interface.....	4
3.2.1	Scaricare la iso del Live CD	4
3.2.2	Masterizzazione dell'immagine iso	5
3.2.3	Avvio	5
3.2.4	Configurazione del Live CD come User Interface.....	5
3.2.4.1	Configurazione hostname	6
3.2.4.2	Configurazione data e ora	6
3.2.4.3	Configurazione software di griglia.....	7
3.2.5	Installazione del Live CD su disco	8
3.3	La User Interface in user space	9
3.3.1	Reperire la UIUS	9
3.3.2	Configurazione della UIUS.....	9
3.3.3	Installazione del software	9
3.3.4	Disinstallazione della UIUS	11
3.3.5	Usare la UIUS per griglie differenti	11
3.4	Installare i certificati	12
3.4.1	Come funzionano i certificati	12
3.4.2	A chi chiedere per avere un certificato personale.....	13
3.4.3	Gestione dei certificati:	13
3.4.4	Generazione e distruzione di un certificato proxy	13
3.4.4.1	Maggiori informazioni sulle estensioni VOMS.....	14
3.4.4.2	Generazione del certificato proxy	15
3.4.4.3	Distruzione del certificato proxy	16
3.4.4.4	Informazioni sul certificato proxy	16
3.4.5	Cambiare password ad un certificato	17
3.4.6	Usare un server MyProxy	17
3.4.6.1	Creazione di un certificato MyProxy non rinnovabile..	18
3.4.6.2	Creazione di un certificato MyProxy rinnovabile	19
3.4.6.3	Informazioni sul certificato MyProxy	20

3.4.6.4	Download di un certificato proxy dal server MyProxy	20
3.4.6.5	Memorizzare pi^81^f9 certificati	21
4	Sottomettere un programma in griglia	22
4.1	Code, paradigmi di esecuzione e file JDL	22
4.1.1	I sistemi di code	22
4.1.2	Il percorso seguito da un job in griglia.	22
4.1.3	Compilazione e linking	23
4.1.4	Tipi di esecuzione	23
4.1.5	I files JDL	23
4.2	Esercizio 1: "Hello World!"	23
4.2.1	Definizione del job	23
4.2.2	Creazione del JDL	24
4.2.3	Sottomissione	24
4.2.4	Controllo dello stato	25
4.2.5	Ritiro	26
4.2.6	Cancellazione	26
4.2.7	In caso di errore... ..	27
4.2.8	Alcuni esempi concreti	27
4.2.8.1	Ottenere informazioni sui Worker Nodes	27
4.2.8.2	Un problema imbarazzantemente parallelo risolto in griglia	29
4.3	Esercizio 2: Monte Carlo	30
4.3.1	Un semplice esempio di Monte Carlo	30
4.3.2	Automatizzazione della sottomissione	31
4.3.2.1	Salvare i job ID in un file	31
4.3.2.2	Uso di una template	32
4.3.2.3	Ancora scripting.	34
4.3.3	Automatizzare il controllo sui job	37
4.3.4	Automatizzare il ritiro dei risultati	38
4.3.5	Linee guida per la completa automatizzazione	41
4.4	Esercizio 3: Monte Carlo in MPI	41
4.4.1	Un esempio MPI	41
4.4.2	Librerie utilizzate	42
4.4.3	JDL per jobs MPI	42
4.5	Esercizio 4: caricare i dati direttamente dalla griglia	44
4.5.1	Input di un job dalla griglia	45
4.5.2	Output di un job dalla griglia	45
4.5.3	Un semplice esempio	46

5	Gestire i file in griglia	49
5.1	Il catalogo LFC - Introduzione	49
5.2	Comandi di uso frequente	50
5.2.1	Ricerca dei file nel catalogo	50
5.2.2	Reperire informazioni sulla struttura della griglia (egrid-infosites)	50
5.2.3	Reperire informazioni sulla struttura della griglia (lcg-infosites)	52
5.2.4	Copiare file locali sul filesystem di griglia	52
5.2.5	Prelevare file dalla griglia sul computer locale	53
5.2.6	Rimozione di file	53
5.2.7	Rinomina di file	54
5.2.8	Creazione e rimozione di directory	54
5.2.9	Creare più nomi logici per il medesimo file	55
5.3	Comandi di uso meno frequente	56
5.3.1	Reperire informazioni relative ai file	56
5.3.1.1	Reperire informazioni sulle Access List	56
5.3.1.2	Recupera il GUID di un file	57
5.3.2	Creare e gestire le repliche	57
5.3.2.1	Visualizza le repliche di un file	58
5.3.2.2	Visualizza gli alias di un file	59
Appendice A	Codice degli esercizi di sottomissione	60
A.1	area.c	60
A.2	areaMPI.c	61
Appendice B	Referenze	65
Indice		66
Indice dei comandi		69

1 Scopo di questo manuale

Scopo di questo tutorial è introdurre gradualmente l'utente ad un uso vantaggioso degli strumenti messi a disposizione da **GRID@Trieste**.

Questa non è una guida di riferimento ai comandi di autenticazione, sottomissione e salvataggio in una griglia, ma un tutorial per l'utente che desideri un'introduzione veloce all'uso di **GRID@Trieste** ed è stata concepita per l'esecuzione passo passo degli esempi presentati.

Prerequisiti per la lettura sono una conoscenza superficiale di linux (quali l'editing di un file, l'uso di base della shell, esecuzione di programmi) e concetti di base di networking.

Poichè sia i comandi e sia l'output degli stessi è spesso molto ingombrante, dove questo rendeva impossibile una formattazione adeguata del testo si è fatto ricorso alle sequenze di caratteri `{\}` e `(...)` per denotare un *line break* inserito per esigenze di stampa e `\\` per indicare la rimozione di una parte dell'output.

2 Concetti di base.

La griglia (GRID) è un paradigma di calcolo per l'utilizzo di risorse computazionali distribuite: l'utente accede a delle risorse senza curarsi di dove queste si trovino e le usa per il salvataggio dei suoi files e per l'esecuzione dei suoi programmi: il software di griglia si occupa al posto suo di gestire lo spostamento dei files, la sicurezza degli spostamenti e il ritiro dei risultati.

La possibilità di includere altri ricercatori in un gruppo e di concedere loro la lettura o la scrittura dei propri files in griglia permette inoltre un'immediata ed efficace condivisione delle risorse e facilita la collaborazione tra utenti diversi anche in diverse parti del mondo.

La tecnologia di griglia consente di avere macchine che erogano servizi di calcolo e di spazio disco, distribuite a livello geografico; la flessibilità introdotta dall'automatismo permette di attingere a più risorse di calcolo e di disco all'aumentare del fabbisogno.

2.1 Una breve panoramica

Fisicamente la griglia è un insieme di computer con sistema operativo GNU/LINUX sui quali è stato installato del software che si occupa di farli funzionare come un sistema e di eseguire programmi dell'utente senza che questo debba preoccuparsi di sceglierne uno o di stabilirvi una connessione direttamente.

Il software usato è quello di LCG, sigla che sta per *LHC Computing Grid*, mentre il live CD¹ (che viene mostrato durante la preparazione dell'ambiente di lavoro, ma che contiene anche il software di installazione per alcune componenti di griglia) è una produzione nata all'interno del progetto **Egrid**.

Attualmente (agosto 2005) **GRID@Trieste** è composta da circa 50 computers non omogenei situati in diversi dipartimenti e organizzazioni di Trieste.

2.2 Componenti principali

Non è nello scopo di questo documento entrare nei particolari della struttura della griglia, ciononostante ci sono alcune nozioni che anche l'utente che voglia utilizzarla come strumento non può ignorare.

2.2.1 Interazione con l'utente

Il primo elemento della griglia visibile è il software installato sulla macchina dell'utente, chiamato anche *middleware*), e cioè l'UI che sta per *User Interface*, che ha il compito di ricevere le richieste dell'utente e sottoporle alle componenti di griglia remote.

Pur essendo considerato uno dei componenti di griglia, la UI non attiva dei servizi sulla macchina locale (non sta in ascolto su alcuna porta) ma rappresenta il lato "client" e quindi ha solo le librerie e i comandi per lavorare e va configurata dall'utente con gli indirizzi dei servizi della griglia. Vedi [Sezione 3.2.4.3 \[Configurazione EGRID\]](#), [pagina 7](#).

¹ cioè un CDROM contenente un sistema operativo e del software che può essere utilizzato senza installazione, semplicemente facendo il boot da CD (da qui il termine "live").

2.2.2 Autenticazione e sicurezza

Per garantire la propria identità e poter accedere alle risorse di griglia, ogni utente è fornito di un *Certificato*: una coppia di file che contengono i propri dati (chiamati *Certificate Subject*) e ne attestano la veridicità.

Attraverso questi file l'utente può creare una chiave (chiamata *Proxy*) che gli permetterà di accedere per un determinato lasso di tempo alle risorse della griglia. Vedi [Sezione 3.4.1 \[teoria\]](#), pagina 12.

Inoltre ogni utente fa parte di una o più *Virtual Organizations* (o VO): queste sono l'analogo dei gruppi di utenti nei sistemi operativi di tipo UNIX e vengono usate (per esempio) per unire utenti dello stesso dipartimento o struttura di ricerca e inibire l'accesso a risorse della griglia a determinati utenti.

2.2.3 Meccanismo di funzionamento

La UI si collega direttamente a due server: il *Resource Broker* (o RB) che gestisce la sottomissione e il *Replica Location Service* (o RLS) che gestisce lo storage in griglia.

Questi si collegano tramite un *Computing Element* ai *Worker Nodes* (che forniscono la potenza computazionale) e agli *Storage Elements* (che permettono il salvataggio dei dati) che sono le risorse di griglia dalle quali attingere.

L'architettura della griglia è in realtà molto più complicata di quanto delineato sopra e ci sono molti dettagli che qui sono stati volutamente trascurati: ulteriori informazioni possono essere reperite nel capitolo relativo la sottomissione (vedi [Sezione 4.1.2 \[Il meccanismo della sottomissione\]](#), pagina 22) e in quello sulla gestione dei dati (**FIXME** [pxref{qualcosa}](#)).

2.2.4 Il sistema informativo interno

La griglia si basa su tre *cataloghi* che raccolgono i dati sui CE (e WN sottostanti) e gli SE: quali sistemi di code vengono usati, l'hardware presente, la possibilità di lanciare job paralleli e cos' via.

Ciascun SE pubblica le proprie caratteristiche su un proprio catalogo, il *Grid Resource Information Service* o GRIS; per il CE le informazioni si trovano sul *Grid Index Information Service* o GIIS; le informazioni dei precedenti vengono infine raccolte in un catalogo principale, il *Berkeley Database Information Index* o BDII.

3 Preparazione dell'ambiente di lavoro (User Interface)

Per poter accedere ai servizi della griglia vengono fornite due soluzioni distinte: è possibile installare il software necessario (*User Interface in User Space*) su un computer GNU/LINUX, anche come utente semplice (vedi [Sezione 3.3 \[UI in user space\]](#), pagina 9), oppure è possibile scaricare ed eventualmente installare su computer un **Live CD** che contiene un sistema operativo completo corredato del software necessario (vedi [Sezione 3.2 \[LiveCD\]](#), pagina 4).

3.1 Requisiti minimi

L'hardware necessario a lavorare in griglia dipende dal tipo di installazione desiderata: è comunque consigliato almeno un Pentium II a 400 Mhz e 128 Mb di Ram per l'esecuzione della UI in userspace oltre a 250 Mb di spazio su disco fisso per l'installazione, mentre per il **Live CD** i requisiti sono gli stessi necessari all'esecuzione e (eventualmente installazione) di un live CD *Knoppix*.

3.2 Il Live CD di EGRID come User Interface

Il **Live CD** di **Egrid** è un CD autoavviante che non richiede installazione e che contiene tutto il software necessario per poter interagire con la griglia nel ruolo di *User Interface*, *Computing Element*, *Worker Node* o *SuperNode*. In questa sezione vedremo come utilizzarlo come semplice UI.

Il **Live CD** è basato sulla distribuzione live *KNOPPIX*¹ e può essere usato da cd oppure installato su disco.

3.2.1 Scaricare la iso del Live CD

Il **Live CD** viene distribuito come una immagine ISO da masterizzare su un CD. Per scaricare l'immagine, dalle dimensioni approssimative di 550Mb, è necessario collegarsi alla pagina <http://www.egrid.it/download/software/SN-download.html> che si presenta più o meno così:

```
Egrid Download SN Registration form
Please provide a valid email address for receiving
the username and password needed for accessing the download area.
```

```
-----
[ Reset ] [ Submit ]
```

Nell'apposito spazio è necessario inserire un indirizzo di posta elettronica valido. Una volta sottomessa la form verrà inviata a tale indirizzo una mail:

```
Egrid Download SN
http://www.egrid.it/download/software/sn
your username is: indirizzo.di@posta.elettronica.valido
your password is: #unaqualchepassword
```

Collegandosi quindi alla pagina <http://www.egrid.it/download/software/sn> e inserendo username e password ricevuti per email sarà possibile accedere alla pagina del download.

¹ che può essere reperita <http://www.knopper.net/knoppix/index-en.html>

Tale pagina conterrà numerosi file chiamati 'egrid-livecd_1.2.7.iso' e 'egrid-livecd_1.2.7.iso.md5'. Il primo contiene l'immagine vera e propria, mentre l'altro contiene il checksum MD5 che servirà per verificare l'integrità del file. I numeri si riferiscono alla versione, quindi vi raccomandiamo di scaricare i file con numero massimo.

3.2.2 Masterizzazione dell'immagine iso

Prima di masterizzare il CD è consigliabile verificarne l'integrità. Sotto LINUX il comando è 'md5sum':

```
bash$ md5sum -vc egrid-livecd_1.2.7.iso.md5
egrid-livecd_1.2.7.iso OK
```

Ovviamente nel caso il risultato sia differente da *OK* dovreste scaricare nuovamente l'immagine ISO.

Una volta verificata l'immagine è sufficiente masterizzare. Sotto LINUX, ammettendo di avere configurato cdrecord, basta il comando

```
bash$ cdrecord egrid-livecd_1.2.7.iso
```

oppure, se usate il programma **k3b**, basterà selezionare dal menu *Tools* il sottomenu 'CD' e poi 'Burn CD Image...'. Da qua selezionate il file 'evrid-livecd_1.2.7.iso' e poi avviate la masterizzazione.

3.2.3 Avvio

Il **Live CD** è perfettamente autonomo. Non installa alcunché sul computer e per farlo partire è sufficiente riavviare il computer con il cd inserito assicurandosi che questo faccia il boot da CD. Consultate il manuale del vostro computer per modificare la sequenza di avvio nel caso fosse necessario.

Durante il boot dovrebbero essere riconosciute tutte le periferiche correttamente. Nel caso alcune periferiche non vengano riconosciute fate riferimento alla documentazione della distribuzione **KNOPPIX** (<http://www.knopper.net/knoppix/index-en.html>) per risolvere tali problemi.

Nel caso vogliate configurare la tastiera e la lingua italiana al boot dovreste scrivere

```
knoppix lang=it
```

3.2.4 Configurazione del Live CD come User Interface

Una volta avviato il **Live CD** è necessario configurare il software di griglia per operare come UI. Per i primi tre passi scritti nel seguito sarà necessario essere root, perciò dovreste aprire un terminale e digitare:

```
knoppix@1[knoppix]$ su -
```

Inoltre perché il software di griglia funzioni sarà necessario assicurarsi che le seguenti porte siano aperte:

host	porte	servizi
egrid-2.egrid.it	2170	ldap BDII
	2811	gridftp
	7512	myproxy
	7772	edg-wl-ns_daemon

	8080	
	8443	
	9000–9001	edg-wl-bkserverd
	9002	edg-wl-logd
host	porte	servizi
..*.*	2119	
	2811	gridftp
	8088	
	20000–25000	gahp_server

3.2.4.1 Configurazione hostname

Perché il software funzioni correttamente è necessario avere la rete correttamente configurata. Nel caso la LAN al quale siete collegati abbia un server DHCP dovreste avere l'indirizzo IP già correttamente configurato, altrimenti da una shell di root date il comando

```
root@1[~]# netcardconfig
```

e seguite le istruzioni a schermo per configurare la rete. Avrete bisogno di conoscere:

- IP address (e.g. 1.2.3.4)
- netmask (e.g. 255.255.255.0)
- subnet (e.g. 1.2.3.255)
- default gateway (e.g. 1.2.3.1)
- ip del o dei nameserver (e.g. 1.2.3.2)

Modificate i files ‘/etc/hosts’ ed ‘/etc/hostname’ e assicuratevi che il vostro *hostname* sia valido. Eventualmente configuratelo con

```
root@1[~]# hostname un.qualche.hostname.valido
```

ATTENZIONE: questa modifica non è temporanea. Nel caso abbiate installato il **Live CD** sul computer dovreste anche modificare il file ‘/etc/hostname’ inserendo l’hostname corretto, ed il file ‘/etc/hosts’ sostituendo ‘Knoppix’ con il vostro hostname

3.2.4.2 Configurazione data e ora

Il live CD sincronizza la data e l’ora tramite il comando **ntpdate** in fase di boot (qualora la rete venga automaticamente configurata in maniera corretta e la porta 123 sia aperta) e quindi in questo caso non è necessario modificarla (anche se verrà mostrata in formato EDT e quindi potrà sembrare non corretta).

In caso contrario² usate il comando

```
root@1[~]# tzconfig
```

e seguite le istruzioni a schermo oppure sincronizzatevi con un server ntp *dopo* aver configurato la rete manualmente:

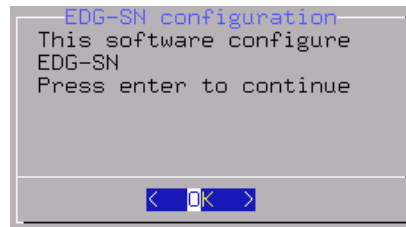
```
root@1[~]# ntpdate pool.ntp.org
```

² quando anche l’indicazione dei minuti dovesse essere scorretta.

3.2.4.3 Configurazione software di griglia

Per configurare il software di griglia lanciate il comando

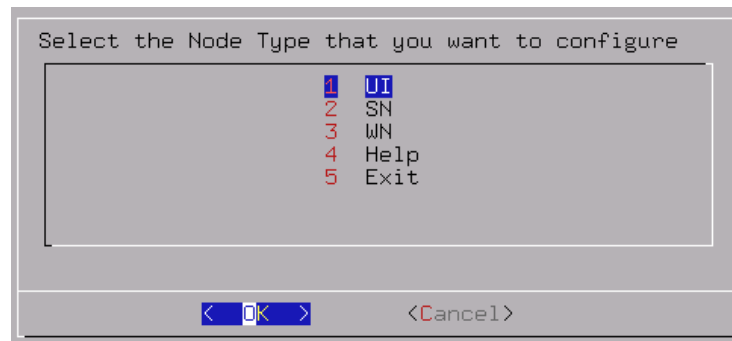
```
root@01[~]# egrid-conf
```



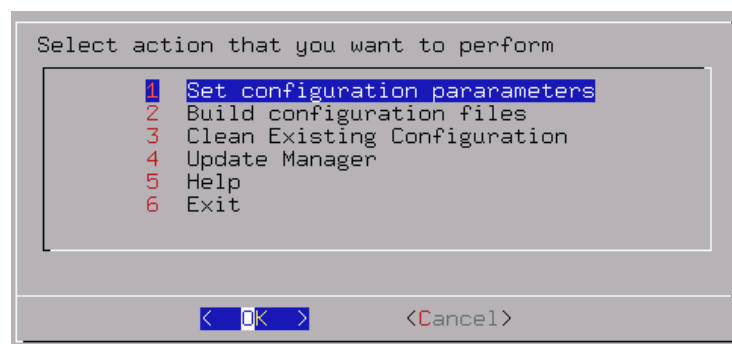
Comparirà un menu di configurazione. Per spostarsi tra le varie opzioni usate le frecce su/giù; per spostarvi tra i pulsanti usate le frecce destra/sinistra. Per selezionare usate il tasto invio.

All'interno del menu le scelte da fare sono le seguenti:

- Selezionare: 'UI'

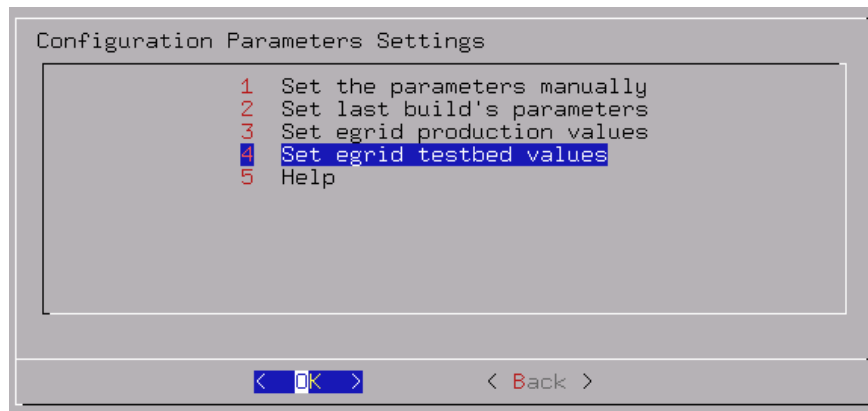


- Dare: 'OK'
- Selezionare: 'Set Configuration Paramaters'

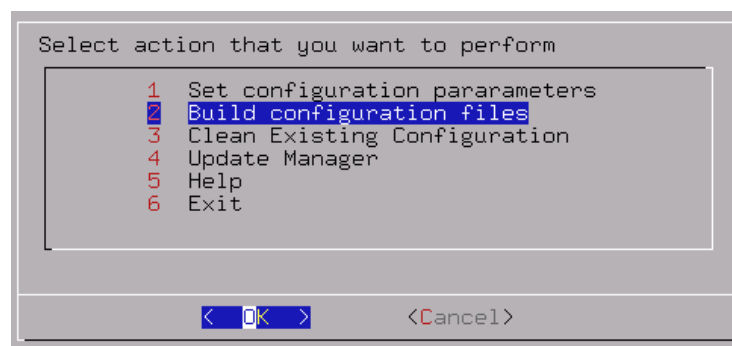


- Dare: 'OK'

- Selezionare: 'Set egrid testbed vaules'



- Dare: 'OK'
- Dare: Back
- Selezionare: 'Build Configuration-files'



- Dare: 'OK'
- Dare: 'Cancel'
- Dare: 'Exit'

A questo punto la configurazione dovrebbe essere stata ultimata.

3.2.5 Installazione del Live CD su disco

Il **Live CD** può essere installato su disco fisso in modo da non dover ripetere ad ogni riavvio la configurazione del sistema; essendo però tale operazione troppo complicata per poter essere descitta brevemente in questo manuale (almeno per i principianti dell'ambiente LINUX) e necessitando di operazioni potenzialmente dannose per i dati pre installati su un computer (come il ripartizionamento del disco) si rimanda per informazioni alla documentazione presente nel **Live CD**.

3.3 La User Interface in user space

La *User Interface in User Space* (UIUS) è un pacchetto software che può essere installato come utente semplice su una macchina GNU/LINUX e fornisce tutto il software necessario ad accedere alla griglia.

3.3.1 Reperire la UIUS

È possibile scaricare la User Interface dal sito di **Egrid**, alla pagina <http://www.egrid.it/download/software/>. I file da scaricare sono quelli del tipo 'egrid-ui_YYYY-MM-DD.tgz' ed è consigliabile scaricare quello con la data più recente.

Attualmente (Settembre 2005) l'ultima versione si trova alla pagina

http://www.egrid.it/download/software/ui/egrid-ui_2005-08-31.tgz

3.3.2 Configurazione della UIUS

Una volta scaricato il file andrà decompresso:

```
bash$ tar xzf egrid-ui_2005-08-31.tgz
```

A questo punto si prospettano due possibilità: mantenere la struttura delle directory invariata, o copiarsi ovunque vogliate il software utile, contenuto nella directory 'egrid-ui_2005-08-31/sw/'. Se volete copiare il software potete semplicemente dare:

```
bash$ mv egrid-ui_2005-08-31/sw/ ~/UI
```

In tal caso in fase di installazione della UIUS dovreste però impostare la variabile *EGRID_INSTALL_DIR* in modo che contenga il percorso completo della directory nella quale avete copiato il software, quindi nell'esempio precedente sarà:

```
bash$ echo $EGRID_INSTALL_DIR
/home/crossi/UI
```

Il file di configurazione predefinito è contenuto nella sottodirectory 'config' del pacchetto scaricato e si chiama 'install.conf'. Nel caso debba essere modificato è possibile usare un semplice editor di testi. Il file predefinito è configurato per accedere al testbed di **Egrid**, usando *egrid* come *Virtual Organization*. Probabilmente sarà necessario apportare un'unica modifica alla VO, sostituendo *gridats* a *egrid* nel file 'config/install.conf', variabile *VO_NAME*.

3.3.3 Installazione del software

Per installare usando le impostazioni di default contenute nel file 'config/install.conf' sarà sufficiente entrare nella directory e dare il comando './install'. L'output apparirà più o meno così:

```
bash$ cd egrid-ui_2005-08-31
bash$ ./install
WARNING: The EDG User Interface tools need a Python interpreter
version 2.2, while your system has 2.3.5 (#2, Feb  9 2005, 00:38:15)
[GCC 3.3.5 (Debian 1:3.3.5-8)]. Please install Python
2.2, or some EDG programs may not function properly.
```

```
WARNING: The EDG User Interface tools need a Java VM version 1.4,
while your system has java version "1.5.0.04". Please install Java 1.4,
or some EDG programs may not function properly.
```

Values to be substituted into templates:

```
<install-dir> := /home/crossi/egrid-ui_2005-08-31/sw
<log-dir> := /home/crossi/egrid-ui_2005-08-31/sw/log
<job-output-dir> := /home/crossi/egrid-ui_2005-08-31/sw/joboutput
<VO-name> := egrid
<BDII-hostname> := egrid-2.egrid.it
<RB-hostname> := egrid-2.egrid.it
<SE-hostname> := egrid-3.egrid.it
<CE-hostname> := egrid-3.egrid.it
<cache-hostname> := egrid-3.egrid.it
<MyProxy-hostname> := egrid-2.egrid.it
<dns-domain> := egrid.it
<lfc-host> := egrid-2.egrid.it
```

Please check that the above values are correct for your setup, then press key 'y' to proceed, or key 'n' to stop now and correct any misconfigurations.

Proceed? (Y/n)

```
creating globus-sh-tools-vars.sh
creating globus-script-initializer
creating Globus::Core::Paths
checking globus-hostname
Done
```

(...)

Values to be substituted:

CONF.GCC	=	_gcc3_2_2
DEFAULT.CE	=	egrid-3.egrid.it
DEFAULT.SE	=	egrid-3.egrid.it
EDG.LOCATION	=	/home/crossi/egrid-ui_2005-08-31/sw/opt/edg
IGNORE.PREFIX	=	true
INFOSERVICE	=	MDS
LOCALDOMAIN	=	egrid.it
MDS.HOST	=	egrid-2.egrid.it
MDS.PORT	=	2170
RLS.MODE	=	LrcOnly
ROS.FAILURE	=	false
STUBFILE	=	

edg-replica-manager-configure uses the following settings:

```
EDG_LOCATION      /home/crossi/egrid-ui_2005-08-31/sw/opt/edg
EDG_LOCATION_VAR  /opt/edg/var
EDG_TMP           /tmp
Program name      /home/crossi/egrid-ui_2005-08-31/sw/opt/edg//sbin/edg-replica-manager-configure
Values file       /home/crossi/egrid-ui_2005-08-31/sw/etc/egrid/edg-replica-manager.conf.values.egrid
```

Files being configured:

```
/home/crossi/egrid-ui_2005-08-31/sw/opt/edg/etc/edg-replica-manager/edg-replica-manager.conf.template
```

EDG Replica Manager configuration log copied to /tmp/edgui-LEyPYc/globus-initialization.log
 Configuring crontab to run edg-fetch-crl-cron every 6 hours...done

In order to use the LCG2/EDG User Interface tools,
 your environment variables need to be properly
 set up. If your shell is sh, bash, or ksh, you
 just need to add the following 2 lines

```
# EGRID/LCG2 User Interface
. /home/crossi/egrid-ui-2005-08-31/sw/etc/egrid/userenv.sh
```

to the shell startup files:

```
/home/crossi/.bashrc
```

Do you want me to add the above lines to '/home/crossi/.bashrc'? (Y/n)

In order to use EGRID, you should put your PEM-encoded
 user key and certificate in the files:

```
/home/crossi/.globus/userkey.pem
/home/crossi/.globus/usercert.pem
```

LCG/EDG UI successfully configured.

Verranno visualizzati un po' di messaggi di errore che in realtà sono normalissimi. Alla fine dello script viene chiesto se si desidera che venga modificato il proprio file `‘.bashrc’` in modo che al login venga caricata automaticamente la configurazione. Altrimenti sarà necessario dare il comando

```
. /path_della_UIUS/etc/egrid/userenv.sh
```

3.3.4 Disinstallazione della UIUS

Per installare la UIUS è sufficiente cancellare la directory contenente la UIUS ed eliminare le eventuali due righe che il software di installazione ha aggiunto al vostro `‘.bashrc’`, identificate dal commento seguente:

```
# EGRID/LCG2 User Interface
```

3.3.5 Usare la UIUS per griglie differenti

La UIUS è corredata di due file di configurazione: il file `‘install.conf’` contiene la configurazione per il testbed di **Egrid**, mentre il file `‘production.conf’` contiene i dati per la griglia di produzione. È possibile creare altri file di configurazione sulla falsa riga dei precedenti e memorizzarli nella directory `‘config’`. In fase di installazione si può scegliere la configurazione da usare tramite l'opzione `‘-c’` seguita dal nome del file eccetto l'estensione `‘.conf’`. Quindi per usare la griglia di produzione il comando sarà:

```
bash$ ./install -c production
```

È possibile eseguire più installazioni in successione ed avere la UIUS configurata per più griglie differenti. In questo caso per poter passare da una configurazione all'altra si usa il comando `‘change-config’`. Senza opzioni esso mostrerà le configurazioni installate e quella attualmente utilizzata:

```
bash$ ./change-config
```

```
./change-config need an argument.
Configuration available:
    install
    production
Actual configuration: production
```

Per passare da una configurazione all'altra sarà sufficiente specificare il nome della configurazione da usare:

```
bash$ ./change-config install
bash$ ./change-config
./change-config need an argument.
Configuration available:
    install
    production
Actual configuration: install
```

Qualora si aggiunga una nuova configurazione o si passi da una all'altra, affinché le modifiche siano valide sarà sufficiente effettuare un nuovo login, o dare i comandi:

```
bash$ . /path_della_UIUS/etc/egrid/unsetenv.sh
bash$ . /path_della_UIUS/etc/egrid/userenv.sh
```

3.4 Installare i certificati

L'identità di un utente in griglia, determinata da un certificato digitale, il cui funzionamento analogo a quello delle chiavi GPG o PGP usate per la firma digitale e per la crittografia. Usando la crittografia a chiave pubblica tale certificato permette di garantire l'identità di un utente.

Per sfruttare le risorse della griglia non si usa direttamente il proprio certificato personale, ma, a partire dal certificato personale, si genera un certificato "sostituto" (*certificato proxy*) che ha una durata molto limitata nel tempo.

3.4.1 Come funzionano i certificati

Ogni utente possiede un *certificato* ed una *chiave privata*, memorizzati rispettivamente in file 'usercert.pem' e 'userkey.pem'. La chiave deve essere conservata gelosamente e mantenuta segreta, perché rappresenta l'identità digitale dell'utente. In caso si dubiti dell'affidabilità della chiave (si smarrita, la macchina sulla quale era conservata stata compromessa o altro) si dovrà contattare la *Certification Authority* perché provveda alla revoca del certificato. Una volta revocato un certificato sarà inutilizzabile e sarà necessario generarne un altro.

Perché una chiave possa essere usata in griglia è necessario che venga firmata da una *Certification Authority* (CA). In questo modo è possibile verificare se una certa chiave è valida o meno, dal momento che sarà sufficiente verificarne la firma digitale. Usando la propria chiave è possibile quindi firmare un qualsiasi documento digitale, in modo che chiunque possa verificare la provenienza del documento stesso: verificando prima che sia stata firmata con la chiave dell'utente, e poi che la chiave dell'utente sia stata firmata dalla CA.

Per aumentare il livello di sicurezza la chiave privata è protetta da una passphrase, che deve anch'essa essere mantenuta segreta.

3.4.2 A chi chiedere per avere un certificato personale

Il certificato viene richiesto dall'utente presso la propria *Registration Authority* (RA) sita presso l'istituzione di appartenenza. Una volta verificata l'autenticità dell'identità dell'utente la RA provveder a inoltrare la richiesta presso la CA, che in caso di approvazione produrrà una firma digitale del certificato stesso.

Alla fine della procedura di registrazione l'utente otterrà due file: il file che contiene la *chiave privata* protetta da una passphrase, `'userkey.pem'`, e quello che contiene la firma della CA, `'usercert.pem'`. Tali file andranno memorizzati sul computer dalla quale si accederà alle risorse di griglia.

Ogni volta che si utilizza il certificato è necessario avere la chiave privata e conoscere la passphrase della stessa.

3.4.3 Gestione dei certificati:

I file contenenti la chiave ed il certificato andranno memorizzati sulla User Interface. Dovranno essere memorizzati nella directory `'./globus'` della propria home con permessi di sola lettura per l'utente proprietario. Nel caso questa directory non esista ed i due file siano conservati nella propria home i comandi da dare saranno:

```
<<<<<<< .mine
bash> mkdir .globus
bash> chmod 700 .globus
bash> mv userkey.pem usercert.pem .globus
bash> cd .globus
bash> chmod 400 userkey.pem usercert.pem
=====
bash$ mkdir .globus
bash$ chmod 700 .globus
bash$ mv userkey.pem usercert.pem .globus
bash$ cd .globus
bash$ chmod 400 userkey.pem usercert.pem
>>>>>>> .r1259
```

3.4.4 Generazione e distruzione di un certificato proxy

Per poter accedere alle risorse di griglia non si usa direttamente la propria chiave ma si genera un certificato temporaneo di durata limitata; in questo modo, il certificato inutilizzabile per altri scopi e dopo la sua scadenza. Un tale certificato si chiama *Certificato Proxy*. Inoltre in questo modo la passphrase della chiave privata verrà richiesta solamente al momento della creazione del certificato proxy.

Il certificato proxy verrà di norma memorizzato sul computer della macchina locale in `'/tmp/x509up_uid'`, dove l'*uid* è quello dell'utente che ha creato il certificato. Tale file è sufficiente ad accedere alle risorse di griglia per il periodo di validità dello stesso (quindi possibile spostarlo su un altro computer, previa rinomina in modo da mantenere coerente la parte finale del nome del file).

Il certificato proxy viene usato per accedere a quasi ogni risorsa della griglia, quindi è necessario anche per l'esecuzione del job. Questo significa che la durata del certificato deve coprire tutta l'esecuzione del job che si sottomesso.

Esistono due tipi di certificati proxy: quelli estesi (*VOMS proxy*) ed i certificati proxy standard (*Globus proxy*). I certificati proxy standard garantiscono solo l'identità dell'utente che li ha generati; invece un certificato proxy VOMS pu riportare informazioni aggiuntive, quali l'appartenenza ad una VO e a (sotto)gruppi di questa, ed anche eventuali ruoli di cui l'utente investito. Useremo qui sempre i certificati proxy estesi VOMS; per ogni scopo pratico sono equivalenti ai certificati proxy standard, quindi non c'è ragione di usare la versione limitata.

3.4.4.1 Maggiori informazioni sulle estensioni VOMS

Questa sezione pu essere omessa in prima lettura.

Il sistema VOMS permette appunto di registrare un un certificato proxy informazioni addizionali su un utente, quali le VO di appartenenza, eventuali combinazioni di gruppi e ruoli all'interno di questa, ed altre informazioni non strutturate ("capabilities"). Queste informazioni addizionali vengono chiamate *attributi* o *estensioni VOMS*.

Ci sono 4 differenti classi di informazioni dentro un singolo attributo VOMS:

VO	Il nome di una VO, p.es. <code>'gridats'</code>
gruppo	Il nome di un gruppo; i gruppi hanno una struttura gerarchica: un gruppo pu avere sottogruppi, e questi a loro volta possono avere sotto-sottogruppi, e cos via. La rappresentazione del nome di un gruppo usata in VOMS usa il carattere <code>"/"</code> come separatore tra gruppi e sottogruppi: p.es., <code>'/gridats/sissa'</code> indica il sottogruppo del gruppo <code>'/gridats'</code> , formato dalle persone afferenti alla SISSA.
ruolo	Un utente pu potenzialmente assumere diversi ruoli (p.es., un amministratore potrebbe anche voler sottomettere propri job come utente normale.) A differenza dei gruppi, i ruoli non sono gerarchici.
capabilities	Le "capabilities" sono arbitrarie stringhe di testo che un amministratore di VO pu associare ad un utente; non sono attualmente in uso, ma riservate per estensioni future.

Una quadrupla (VO,gruppo,ruolo,capability) prende il nome di *FQAN*, ovvero "Fully Qualified Attribute Name". Nella rappresentazione di un FQAN, si usa il carattere `"/"` per separare i vari elementi, e la stringa `'NULL'` per indicare l'assenza di ruoli o capability specifiche; p.es., `'/gridats/Role=NULL/Capability=NULL'` indica lo FQAN corrispondente al gruppo `'/gridats'` della VO `'gridats'`, nessun ruolo e nessuna capability assegnati.

Ogni VO ha il proprio server VOMS, dove gli amministratori della VO possono definire gruppi e ruoli, e popolarli di utenti. Ogni VO ha sempre almeno un gruppo, chiamato come la VO stessa, che raccoglie tutti gli utenti di quella VO; p.es., la VO `'gridats'` ha il gruppo `'/gridats'`.

Le estensioni VOMS sono registrate nel certificato richiesto da un utente sotto forma di lista di FQAN. Al momento della generazione del certificato proxy, il programma `voms-proxy-init` contatta il server VOMS di una certa VO per ottenere la lista degli FQAN associati ad un utente; per ogni combinazione gruppo/ruolo che l'utente pu assumere, viene inserito il corrispondente FQAN nel certificato proxy.

3.4.4.2 Generazione del certificato proxy

Per *generare* un certificato proxy si usa il comando `voms-proxy-init`:

```
voms-proxy-init [-valid h:m] [-cert certfile] \
  [-key keyfile] [-certdir certdir] [-out proxyfile] \
  [-voms vo] [-order gruppo[:ruolo]]
```

dove:

‘-valid h:m’

specifica la durata della validit  del certificato proxy. Se l’opzione   omessa la durata sar  di 12 ore.

‘-cert certfile’

Specifica un nome alternativo per il file contenente il certificato. Il nome di default   ‘\$HOME/.globus/usercert.pem’, dove \$HOME   la home directory dell’utente

‘-key keyfile’

Specifica un nome alternativo per il file contenente la chiave dell’utente. Il nome di default   ‘\$HOME/.globus/userkey.pem’, dove \$HOME   la home directory dell’utente

‘-certdir certdir’

Specifica un nome alternativo per la directory contenente il certificato e la chiave dell’utente. Il nome di default   ‘\$HOME/.globus/’, dove \$HOME   la home directory dell’utente

‘-out proxyfile’

Specifica un nome di file alternativo per il certificato proxy. Il nome di default   ‘/tmp/x509up_ uid’ con  uid l’user id sulla macchina.

‘-voms vo’

Specifica il nome della VO di appartenenza; verr  contattato un server (uno specifico per ogni VO), da cui ottenere l’elenco dei gruppi e dei ruoli a cui l’utente appartiene, e questo elenco viene riportato all’interno del certificato. Se questa opzione   omessa, sar  generata una proxy standard, priva delle estensioni VOMS.

‘-order gruppo[:ruolo]’

Permette di includere nelle estensioni VOMS solo alcuni tra i gruppi a cui si appartiene, oppure solo alcuni tra i ruoli. Pu  essere ripetuta pi  volte, con effetti cumulativi. Se viene omessa, tutti le combinazioni gruppo/ruolo a cui un utente appartiene saranno riportate nel certificato proxy VOMS.

Per creare un certificato proxy per la VO ‘gridats’ della validit  di 24 ore, ad esempio, il comando da dare   il seguente:

```
<<<<<<< .mine
bash> voms-proxy-init -voms gridats
Your identity: /C=IT/O=INFN/OU=Personal Certificate/L=ICTP/CN=Carlo Rossi█
=====
bash$ voms-proxy-init -voms gridats
```

```

Your identity: /O=GRID@Trieste/DC=it/DC=trieste/DC=esempio/CN=Carlo Rossi
>>>>>> .r1259
Enter GRID pass phrase for this identity:
<<<<<<< .mine
Creating temporary proxy ..... Done
/C=IT/O=INFN/OU=Host/L=ICTP/CN=egrid-1.egrid.it
/C=IT/O=INFN/CN=INFN Certification Authority
Creating proxy..... Done
Your proxy is valid until Fri Aug  5 23:18:29 2005
=====
Creating temporary proxy ..... Done
/C=IT/O=INFN/OU=Host/L=ICTP/CN=egrid-1.egrid.it
/C=IT/O=INFN/CN=INFN Certification Authority
Creating proxy..... Done
Your proxy is valid until Tue Oct 18 22:45:50 2005
>>>>>> .r1259

```

Da questo momento fino alla scadenza del certificato proxy tutti i comandi per interagire con la griglia saranno operativi.

3.4.4.3 Distruzione del certificato proxy

Per distruggere un certificato proxy sar sufficiente dare il comando `voms-proxy-destroy`, o cancellare il file `/tmp/x509up_uuid`.

3.4.4.4 Informazioni sul certificato proxy

Per ottenere informazioni sul certificato proxy esiste il comando `<<<<<<< .mine voms-proxy-info`. Ad esempio: `===== voms-proxy-info`. Ad esempio:

```

bash$ voms-proxy-info
VO                : gridats
Valid from        : Oct 18 08:45:47 2005 GMT
Valid to          : Oct 18 20:45:47 2005 GMT
>>>>>> .r1259

<<<<<<< .mine ===== Per ottenere informazioni complete, si usa l'opzione '-all':
>>>>>> .r1259

<<<<<<< .mine
bash> voms-proxy-info
=====
bash$ voms-proxy-info -all
Subject           : /O=GRID@Trieste/DC=it/DC=trieste/DC=esempio/CN=Carlo Rossi/CN=proxy
Issuer            : /O=GRID@Trieste/DC=it/DC=trieste/DC=esempio/CN=Carlo Rossi
Type              : proxy
Bits              : 512
Valid From        : Oct 18 08:40:53 2005 GMT
Validity left     : Oct 18 20:45:53 2005 GMT
VO                : gridats
Holder Subject    : /O=GRID@Trieste/DC=it/DC=trieste/DC=esempio/CN=Carlo Rossi

```

```

Holder CA      : /O=GRID@Trieste/DC=it/DC=egrid/CN=GRID@Trieste CA 1
Issuer Subject : /C=IT/O=INFN/OU=Host/L=ICTP/CN=egrid-1.egrid.it
Issuer CA      : /C=IT/O=INFN/CN=INFN Certification Authority
Valid from     : Oct 18 08:45:47 2005 GMT
Valid to       : Oct 18 20:45:47 2005 GMT
Attribute      : /gridats/Role=NULL/Capability=NULL

```

Come si pu vedere, in questo caso `voms-proxy-init` stampa anche gli attributi in formato FQAN.

Nel caso di una proxy standard (priva di estensioni VOMS), il risultato del comando sar pi breve:

```

bash$ voms-proxy-info -all
>>>>>> .r1259
subject  : /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi/CN=proxy
issuer   : /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
identity : /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
type     : full legacy globus proxy
strength : 512 bits
path     : /tmp/x509up_u666
timeleft : 23:59:59

```

Attenzione! Con una proxy standard, senza estensioni VOMS, il comando `voms-proxy-info` usato senza opzioni non produce alcun risultato!

3.4.5 Cambiare password ad un certificato

Nel caso si desideri cambiare la passphrase della propria chiave personale disponibile il comando `grid-change-pass-phrase`. Esso chieder la vecchia password e successivamente una nuova password.

```

bash$ grid-change-pass-phrase
read RSA key
Enter PEM pass phrase:
writing RSA key
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:

```

3.4.6 Usare un server MyProxy

L'uso dei certificati utente e dei certificati proxy garantisce un elevato livello di sicurezza impedendo ad utenti non autorizzati di accedere a tutte o alcune risorse della griglia, ma come effetto collaterale rende la durata del certificato proxy determinante per alcuni compiti.

In particolare nel caso il certificato proxy scada durante l'esecuzione del job sar impossibile recuperarne l'output o recuperare informazioni sul job stesso. Inoltre sar impossibile accedere alla griglia senza avere i propri certificati personali, che per andrebbero custoditi gelosamente, perci nel caso un utente usi spesso pi computer come UI potrebbero nascere delle falle di sicurezza.

Una soluzione parrebbe aumentare a dismisura la durata del certificato proxy, ma non essendo questo protetto da alcuna password introduce un altro problema di sicurezza: un utente non autorizzato potrebbe riuscire a copiare il proxy ed usarlo per i propri scopi.

La soluzione migliore consiste invece nell'uso del servizio *MyProxy* che ha quindi due funzionalit  distinte:

- conserva i certificati proxy in modo da poter ottenere un certificato proxy valido senza bisogno di avere la chiave utente,
- permette di prolungare la durata di un certificato proxy in maniera automatica, in modo da evitare che questo scada prima che i job sottomessi terminino regolarmente

Per quanto riguarda la prima funzionalit  il meccanismo   il seguente:

- viene creato un certificato MyProxy che viene memorizzato su un server remoto (identificato dalla variabile d'ambiente `MYPROXY_SERVER`), usando la chiave personale dell'utente e protetto da una passphrase. Il certificato memorizzato sul server remoto ha una scadenza di default di 7 giorni.
- il server myproxy tramite questo certificato (firmato con la chiave dell'utente) pu  generare dei certificati proxy validi.
- L'utente pu  a suo piacimento, conoscendo esclusivamente la passphrase del certificato myproxy, ottenere dal server myproxy un certificato valido della durata di 12 ore.

La seconda funzionalit    analoga ed   del tutto trasparente all'utente:

- viene creato un certificato MyProxy che viene memorizzato su un server remoto (identificato dalla variabile d'ambiente `MYPROXY_SERVER`), usando la chiave personale dell'utente e protetto da una passphrase. Il certificato ha una scadenza di default di 7 giorni. Tale certificato *non* sar  protetto da passphrase.
- il server myproxy tramite questo certificato (firmato con la <<<<<<< .mine chiave dell'utente) pu  generare dei certificati proxy validi.
- I servizi che ne hanno necessit  possono richiedere la generazione di un certificato proxy valido al server MyProxy. (L'unico servizio che ha per ora questa capacit    il servizio RB, che rinnova un certificato proxy in scadenza se il job associato sta ancora girando.) Gli utenti invece *non* saranno in grado di ottenere un certificato proxy tramite il certificato MyProxy rinnovabile. ===== chiave dell'utente) pu  generare dei certificati proxy validi.
- I servizi che ne hanno necessit  possono richiedere la generazione di un certificato proxy valido al server MyProxy. (L'unico servizio che ha per ora questa capacit    il servizio RB, che rinnova un certificato proxy in scadenza se il job associato   ancora in esecuzione.) Gli utenti invece *non* saranno in grado di ottenere un certificato proxy tramite il certificato MyProxy rinnovabile. >>>>>>> .r1259

3.4.6.1 Creazione di un certificato MyProxy non rinnovabile

La generazione di un certificato myproxy avviene con il comando `myproxy-init`:

```
bash$ myproxy-init -d -k delegation
Your identity: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Proxy Verify OK
Your proxy is valid until: Wed Aug 31 11:54:25 2005
Enter MyProxy pass phrase: passphrase di almeno 6 caratteri
Verifying password - Enter MyProxy pass phrase:
```

```
A proxy valid for 168 hours (7.0 days) for user      {\}
/C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi {\}
now exists on egrid-2.egrid.it.
```

La durata predefinita di un certificato myproxy di 7 giorni (168 ore). Pu essere specificata un'altra durata usando l'opzione '-c' seguita dal numero di ore richiesto.

L'opzione '-d' specifica di usare il *Distinguished Name* come identificativo del certificato myproxy anzich il nome di login. una buona cosa usare sempre tale opzione.

L'opzione '-k' assegna un nome al particolare certificato. Possono essere memorizzati pi certificati con pi nomi, ma importante ricordare che il *certificato per il rinnovo non deve essere creato con l'opzione '-k'*. Perci si consiglia di usare sempre l'opzione '-k' per certificati myproxy tranne che per i certificati rinnovabili dai servizi.

Per ottenere informazioni sui certificati attualmente memorizzati su un server MyProxy, si usa il comando `myproxy-info`, sempre con l'opzione '-d':

```
bash$ myproxy-info -d
username: /C=IT/O=INFN/OU=Personal Certificate/L=ICTP/CN=Riccardo Murri
owner: /C=IT/O=INFN/OU=Personal Certificate/L=ICTP/CN=Riccardo Murri
name: delegation
timeleft: 167:59:34 (7.0 days)
```

3.4.6.2 Creazione di un certificato MyProxy rinnovabile

Il comando `myproxy-init` come nel caso precedente, ma necessario specificare alcune opzioni. Innanzitutto bisogna decidere chi potr rinnovare il certificato. L'opzione '-A' permette a qualsiasi servizio di farlo, altrimenti si pu limitare questa possibilit ad un particolare host tramite l'opzione '-R'.

Perch il certificato di rinnovo funzioni correttamente necessario non usare l'opzione '-k', perci se si desidera memorizzare anche un certificato myproxy non rinnovabile sar necessario creare il certificato per la delegation con l'opzione '-k', e quello per il rinnovo senza questa opzione. Si noti inoltre che il certificato myproxy rinnovabile *non* pu essere scaricato con il comando `myproxy-get-delegation` come invece avviene per il certificato non rinnovabile.

```
bash$ myproxy-info -d -A
Your identity: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Proxy Verify OK
Your proxy is valid until: Wed Aug 31 14:44:14 2005
A proxy valid for 168 hours (7.0 days) for user      {\}
/C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi {\}
now exists on egrid-2.egrid.it.
```

In questo caso non verr richiesta alcuna passphrase per il nuovo certificato (a parte, ovviamente quella della chiave personale necessaria per generare il certificato myproxy), dal momento che i servizi che richiederanno il rinnovo del certificato non avranno modo di conoscerla. Le opzioni usate sono:

'-d' Usa il Distinguished Name come identificativo del MyProxy. necessaria al fine del buon funzionamento del certificato.

- ‘-A’ Non impone limitazioni sui client che possono rinnovare il certificato. Al posto di ‘-A’ si pu usare ‘-R’
- ‘-R’ Impone che il certificato possa essere rinnovato solamente dai client il cui Common Name (CN) comprende una certa stringa. Ad esempio ‘-R condorg/modi4.ncsa.uiuc.edu’. Se si aggiunge l’opzione ‘-x’ verr considerato il Distinguished Name (DN) anzich solo il CN (ad esempio /C=US/O=National Computational Science Alliance/CN=condorg/modi4.ncsa.uiuc.edu)

Anche in questo caso si pu aumentare la durata di un certificato proxy con l’opzione ‘-c <ore>’, e si pu distruggere un certificato rinnovabile con il comando **myproxy-destroy**, eventualmente seguito dall’opzione ‘-k’ e dal nome che si specificato per il certificato rinnovabile.

3.4.6.3 Informazioni sul certificato MyProxy

Per ottenere informazioni sul certificato myproxy si usi il comando **myproxy-info**:

Nel caso si abbia creato un solo certificato non rinnovabile con opzione ‘-k *delegation*’ il risultato sar:

```
bash$ myproxy-info -d
username: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
owner: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
name: delegation
timeleft: 167:38:02 (7.0 days)
```

Se sono stati creati pi certificati myproxy il risultato potrebbe apparire come:

```
bash$ myproxy-info -d
username: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
owner: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
name: delegation
timeleft: 165:37:47 (6.9 days)
owner: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
name: secondario
timeleft: 477:38:00 (19.9 days)
owner: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
renewal policy: *
timeleft: 167:58:43 (7.0 days)
```

in cui il certificato rinnovabile identificato dalla scritta

```
renewal policy: *
```

Il campo ‘name:’ invece mostra il nome che stato assegnato al certificato al momento della creazione con l’opzione ‘-k’.

3.4.6.4 Download di un certificato proxy dal server MyProxy

Una volta generato il certificato myproxy non rinnovabile sar possibile scaricare un certificato proxy valido con il comando **myproxy-get-delegation**:

```
bash$ myproxy-get-delegation -d -k delegation
Enter MyProxy pass phrase:
A proxy has been received for user {}
```

```
/C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi {}
in /tmp/x509up_u1000
```

A questo punto sar possibile accedere in griglia.

L'opzione '-k' anche in questo caso permette di specificare quale certificato si vuole scaricare. Se un certificato non ha nome pu essere scaricato senza usare l'opzione '-k'.

3.4.6.5 Memorizzare pi certificati

Il server MyProxy permette di memorizzare pi certificati. Per poterli distinguere sar necessario usare l'opzione '-k'. Quindi se si volesse caricare un nuovo certificato della durata di 20 giorni:

```
bash$ myproxy-init -d -c 480 -k secondario
Your identity: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Proxy Verify OK
Your proxy is valid until: Tue Sep 13 12:23:31 2005
Enter MyProxy pass phrase:
Verifying password - Enter MyProxy pass phrase:
A proxy valid for 480 hours (20.0 days) for user {}
/C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi {}
now exists on egrid-2.egrid.it.
```

Il comando myproxy-info ovviamente mostrer le informazioni relative ai due certificati:

```
bash$ myproxy-info -d
username: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
owner: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
name: delegation
timeleft: 167:59:39 (7.0 days)
owner: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
name: secondario
timeleft: 479:59:52 (20.0 days)
owner: /C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi
renewal policy: *
timeleft: 167:58:43 (7.0 days)
```

e se si volesse scaricare un certificato proxy usando il secondo certificato si dovrebbe usare il comando:

```
bash$ myproxy-get-delegation -d -k secondario
Enter MyProxy pass phrase:
A proxy has been received for user {}
/C=IT/L=Trieste/O=EGRID/OU=ICTP/CN=Carlo Rossi {}
in /tmp/x509up_u1000
```

fornendo, ovviamente, la passphrase data durante la generazione del secondo certificato.

4 Sottomettere un programma in griglia

Configurato l'ambiente di lavoro ci occupiamo ora dell'esecuzione in griglia di diversi tipi di programmi: dopo una prima introduzione verranno presentati diversi job “tipo” che illustrano i vari metodi di lavoro.

4.1 Code, paradigmi di esecuzione e file JDL

Configurato l'ambiente di lavoro ci occupiamo ora dell'esecuzione di alcuni programmi: dopo una prima introduzione verranno presentati diversi job “tipo” che illustrano i vari metodi di lavoro.

4.1.1 I sistemi di code

I sistemi di code sono programmi che gestiscono l'esecuzione dei jobs (lo *scheduling*) per distribuire le risorse in maniera equa tra i diversi utenti; il criterio di equità dipende da diversi parametri (occupazione delle risorse, economici, ecc. . .).

I vantaggi di tale sistema è che l'utente non deve preoccuparsi di cercare una CPU libera per eseguire un programma, ma *sottomette* il suo job e il sistema di code si occupa di farlo eseguire quando questa si libera, inoltre questo sistema diminuisce il tempo medio durante il quale le risorse sono inutilizzate.

L'utente ha inoltre la possibilità di controllare in tempo reale lo stato di esecuzione del suo programma.

4.1.2 Il percorso seguito da un job in griglia.

Nell'introduzione è stato delineato velocemente il ruolo del *Resource Broker* come l'unico elemento visibile all'utente (al di fuori del *middleware*) nella sottomissione: questo elemento raccoglie in realtà più servizi responsabili della gestione dei job.

In fase di sottomissione il RB attraverso i servizi che lo compongono accetta le richieste di job (*Network Server*) con i dati che con essa vengono spediti nella *Sandbox* (vedi Sezione 4.2.2 [Creazione del JDL], pagina 24), cerca da database esterni (il BDII¹ e il già citato RLS) un sito adatto all'esecuzione² chiamato *Computing Element* o CE che contiene un sistema di code ed è un gateway verso le risorse di calcolo e di storage.

A questo punto il job sta girando su uno o più (nel caso parallelo) dei *Worker Nodes* (o WN) relative al CE selezionato, cioè su uno dei computer che forniscono la potenza di calcolo e altri componenti del RB si occupano di adattare alla griglia il job, attraverso dei wrappers, creati dal *Job Adapter* (o JA) e sottomessi al CE (che li riceve attraverso un programma chiamato *CondorG*) dal *Job Controller* o JC.

Per controllare lo stato dell'esecuzione, sempre il RB, controlla i messaggi di log di *CondorG* (attraverso il *Log Monitor* o LM) e concentra, raccoglie e mantiene le informazioni ottenute dai vari sistemi (*Logging e Booking Service* o LBS).

Per l'utente tutti questi passaggi avvengono in maniera trasparente e diventano visibili solo ispezionando i messaggi di log relativi a un job sottomesso (vedi Sezione 4.2.7 [In caso di errore...], pagina 27).

¹ *Berkley Database Information Service*: mantiene le informazioni relative alle risorse disponibili in griglia

² questo processo viene chiamato *matchmaking*

4.1.3 Compilazione e linking

Poichè i WN possono non avere installate le stesse librerie della UI, è consigliabile che l'utente faccia il linking statico dei propri programmi (con i compilatori GNU questo significa aggiungere un'opzione `-static` in coda ai comandi di compilazione).

Essi vengono inoltre eseguiti in griglia su macchine del tipo x86 con sistema operativo LINUX e quindi devono essere compilati su macchine dello stesso tipo³.

4.1.4 Tipi di esecuzione

La griglia supporta sia job *seriali* dove ogni macchina esegue un compito indipendentemente dalle altre, che di tipo **MPI** (via le librerie MPICH).

4.1.5 I files JDL

La griglia assegna per l'esecuzione dei nostri job un CE in funzione delle risorse di cui ha bisogno: per scoprire quali sono e per ricevere i dati sulle modalità di esecuzione dei programmi essa ha bisogno di informazioni da parte nostra e queste gli vengono fornite da un file in un formato chiamato *Job Description Language* (o JDL)⁴.

Si tratta di un file di testo nella forma:

```
[  
keyword1  = value1 ;  
keyword2  = value2 ;  
...  
]
```

le *keyword* sono stringhe come 'JobType' o 'Executable' che rappresentano dei parametri di esecuzione o dei requisiti ai quali vengono associati dei valori appropriati.

Il formato del file è *case sensitive*.

4.2 Esercizio 1: “Hello World!”

Con questo esercizio impareremo i meccanismi di base dell'esecuzione in griglia (sottomissione, controllo dello stato, ritiro dei risultati) che qui verranno descritti in dettaglio.

In seguito sono descritte due variazioni che fanno uso delle tecniche viste per eseguire compiti di utilità meno simbolica.

4.2.1 Definizione del job

Scopo del nostro job è di far stampare su un WN la scritta 'Hello World!' in maniera analoga a quanto accade con il comando `echo 'Hello World!'`.

Questo job non ha input e, come output, desideriamo un file contenente lo **stdout** del comando.

³ o con l'aiuto di un *cross-compiler*, cioè di un compilatore che trasforma del codice C in un eseguibile per un'architettura e/o sistema operativo diversi da quelli sui quali è stato eseguito.

⁴ che è quello letto da *CondorG* in fase di sottomissione. Vedi Sezione 4.1.2 [Il meccanismo della sottomissione], pagina 22.

4.2.2 Creazione del JDL

Per l'esecuzione useremo sui *Worker Node* il comando `echo` (cioè `/bin/echo`) con un argomento (`'Hello World!'`); questi nel JDL assumono la forma:

```
Executable = "/bin/echo";
Arguments  = "Hello World!";
```

Ci serve che la griglia salvi lo **stdout** del programma in un file (per esempio `'stdout.txt'`) e quindi aggiungiamo:

```
StdOutput = "stdout.txt";
```

Per lo **stderr** è disponibile una *keyword* analoga, che includiamo:

```
StdError = "stderr.txt";
```

Abbiamo specificato i nomi dei files nei quali salvare l'I/O del programma ma non come ritirarli: per questo scopo useremo la *Sandbox*, un meccanismo per trasferire file di dimensioni modeste (entro i 25 MB in totale) dalla *User Interface* al WN e viceversa.

Ci interessa ricevere la trascrizione degli *stream* del nostro comando e quindi scriviamo:

```
OutputSandbox={"stderr.txt","stdout.txt"};
```

A questo punto il nostro JDL (che salviamo in un file chiamato `'Hello.jdl'`) è pronto e si presenterà così:

```
[
    Executable = "/bin/echo";
    Arguments  = "Hello World!";

    StdOutput = "stdout";
    StdError = "stderr";

    OutputSandbox = {"stdout","stderr"};
]
```

4.2.3 Sottomissione

Dopo aver creato un *Proxy* Vedi [Sezione 3.4 \[Autenticazione\], pagina 12.](#)) sarà sufficiente digitare il comando:

```
bash$ edg-job-submit --vo gridats Hello.jdl
```

per ottenere (usualmente) dopo pochi secondi:

```
Selected Virtual Organisation name (from --vo option): gridats
Connecting to host egrid-2.egrid.it, port 7772
Logging to host egrid-2.egrid.it, port 9002
```

```

*****//*****
                                JOB SUBMIT OUTCOME
The job has been successfully submitted to the Network Server.
Use edg-job-status command to check job current status. \
  Your job identifier (edg_jobId) is:

- https://egrid-2.egrid.it:9000/jfxP6_Rgu15vj26ZR03LvQ

*****//*****

```

Analizziamo l'output del comando.

La prima riga ci informa che la nostra *Virtual Organization* è stata specificata via linea di comando.

Trascurando dettagli inessenziali alla sottomissione, le 2 righe che seguono indicano che il collegamento al *Resource Broker* (attualmente **egrid-2.egrid.it**) è avvenuto in maniera corretta.

La parte che ci interessa è il *job id*:

```
https://egrid-2.egrid.it:9000/jfxP6_Rgu15vj26ZR03LvQ
```

è tramite questo che il nostro programma viene identificato in griglia e perderlo significa non poter ritirare i risultati a esecuzione conclusa.

Qualora il risultato del comando di sottomissione dovesse essere diverso andrà ricontrollato il file `‘.jdl’`, il *proxy* o la configurazione del sistema.

4.2.4 Controllo dello stato

Per vedere lo stato del nostro job digitiamo:

```
bash$ edg-job-status \
      https://egrid-2.egrid.it:9000/jfxP6_Rgu15vj26ZR03LvQ
```

Per ottenere un messaggio simile al seguente:

```

*****
BOOKKEEPING INFORMATION:

Status info for the Job : https://egrid-2.egrid.it:9000/jfxP6_Rgu15vj26ZR03LvQ
Current Status:         Scheduled
Status Reason:          Job successfully submitted to Globus
Destination:            egrid-ce-01.egrid.it:2119/jobmanager-lcgpbs-gridats
reached on:             Tue Aug  9 15:11:19 2005

*****

```

Current Status) mostra lo stato del job (approfondito da *Status Reason*), che può essere:

SUBMITTED, WAITING, READY: in fase di sottomissione: il gestore delle code non è ancora stato raggiunto.
 SCHEDULED: in coda.
 RUNNING: in esecuzione.
 DONE: job eseguito: output disponibile per essere ritirato.
 ABORTED: annullato (per esempio a causa di un proxy scaduto o per timeout causato da esecuzione troppo lunga).
 CANCELLED: cancellato dall'utente.
 CLEARED: l'output del job è stato ritirato.

Sotto *Destination* c'è il nome del gestore delle code (con l'indirizzo completo).

Per controllare lo stato del job a intervalli regolari si può usare il comando `watch` (*man watch*).

```
bash$ watch -n 20 \
    edg-job-status \
    https://egrid-2.egrid.it:9000/jfxP6_Rgu15vj26ZR03LvQ
```

E in questo modo si otterrà ogni 20 secondi la stampa a terminale dello stato.

4.2.5 Ritiro

Quando il job abbia raggiunto lo stato SCHEDULED (questione usualmente di pochi minuti) è solo una questione di tempo che questo venga eseguito su uno dei *Worker Node* (a meno che il Proxy non scada prima).

Aspettiamo che il job venga eseguito (Current status = *Done*) e scriviamo:

```
bash$ edg-job-get-output --dir . \
    https://egrid-2.egrid.it:9000/jfxP6_Rgu15vj26ZR03LvQ
```

Per ritirare l'output. Otteniamo:

```
Retrieving files from host: \
  egrid-2.egrid.it ( for https://egrid-2.egrid.it:9000/8fXRheHe0oWSiUDGYUT7Wg )

*****
                        JOB GET OUTPUT OUTCOME
*****

Output sandbox files for the job:
- https://egrid-2.egrid.it:9000/jfxP6_Rgu15vj26ZR03LvQ
have been successfully retrieved and stored in the directory:
/tmp/testuser_jfxP6_Rgu15vj26ZR03LvQ

*****
```

In questo caso abbiamo deciso che l'output sia salvato nella directory corrente (con l'opzione `--dir`), altrimenti verrebbe messo in una di default.

L'output viene messo in una directory nella forma `'username_hash'` dove *username* è lo username dell'utente e *hash* è la parte terminale del job ID.

In questo caso dentro la directory ci saranno un file `'stderr.txt'` (vuoto) e `'stdout.txt'` con l'output previsto.

4.2.6 Cancellazione

Per cancellare il job prima del ritiro del risultato (e liberare delle risorse) si usa:

```
bash$ edg-job-cancel \
      https://egrid-2.egrid.it:9000/jfxP6_Rgu15vj26ZR03LvQ
```

Per ulteriori informazioni, tutti i comandi del tipo `edg-job-*` hanno un help di linea al quale si accede con l'opzione `--help`.

4.2.7 In caso di errore...

Qualora dopo la sottomissione il job dovesse abortire senza motivo, qualora la causa del problema non sia collegabile all'utente la cosa migliore da fare è avvertire gli amministratori di sistema.

Per farlo in maniera efficace si usa il comando:

```
bash$ edg-job-get-logging-info -v 2 ID_del_job \
      -o logging_file.txt
```

che salva in `logging_file.txt` informazioni utili per il rintracciamento del problema e che va spedito insieme a una copia del file JDL e a una breve descrizione del problema.

4.2.8 Alcuni esempi concreti

Ciò che è stato mostrato finora è già sufficiente allo svolgimento di alcuni compiti utili in griglia.

Qui di seguito presentiamo due esempi pratici che fanno uso delle conoscenze precedentemente descritte.

4.2.8.1 Ottenere informazioni sui Worker Nodes

Vedremo ora un modo per ottenere informazioni sul sistema sul quale sta girando la nostra applicazione.

Per farlo creiamo un semplice script BASH che chiamiamo `info.sh` e lo sottomettiamo.

```
#!/bin/bash

# Vedo dove sta girando il job.
hostname > hostname.txt

# Questo file contiene la quantità di memoria disponibile
cp /proc/meminfo meminfo.txt

# E questo il tipo di cpu.
cp /proc/cpuinfo cpuinfo.txt

# Questa e' la directory dove stiamo lavorando:
pwd > working_dir.txt

# Comprimo i risultati, che verrenno poi ritirati
# poteva essere fatto direttamente sui files in /proc
tar cvjf results.tar.bz2 \
      hostname.txt meminfo.txt cpuinfo.txt working_dir.txt
```

Per semplificare la successiva elaborazione dei dati salviamo i risultati in file differenti e poi li archiviamo con `tar` (*man tar*).

Il JDL (che chiameremo ‘info.jdl’) è simile a ‘HelloWorld.jdl’ e cioè:

```
[
    Executable = "info.sh";
    InputSandbox  = {"info.sh"};
    OutputSandbox = {"results.tar.bz2"};
]
```

È sempre buona norma specificare il ritiro della trascrizione dello **stdout** e **stderr** anche se il loro contenuto non fosse interessante durante l’esecuzione del programma, per controllare eventuali messaggi d’errore qualora l’esecuzione dei job non dovesse andare a buon fine; con tale aggiunta otteniamo:

```
[
    Executable = "script.sh";

    StdOutput = "stdout.txt";
    StdError = "stderr.txt";

    InputSandbox  = {"script.sh"};
    OutputSandbox = {"stdout.txt", "stderr.txt", "results.tar.bz2"};
]
```

Che può essere salvato (per esempio come ‘job.jdl’) nella directory che contiene lo script e successivamente sottomesso, come in precedenza.

Una volta scompattato l’archivio ‘results.tar.bz2’ si ottiene una lista di file con le informazioni desiderate, come per esempio ‘cpuinfo.txt’:

```
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 11
model name    : Intel(R) Pentium(R) III CPU family      1133MHz
stepping      : 1
cpu MHz       : 1128.633
cache size    : 512 KB
physical id   : 0
siblings      : 1
runqueue      : 0
fdiv_bug      : no
...
```

o ‘meminfo.txt’:

```
total:      used:      free:  shared: buffers:  cached:
Mem: 1049833472 988823552 61009920      0 128397312 741892096
Swap: 1073733632      0 1073733632
MemTotal:      1025228 kB
MemFree:        59580 kB
MemShared:        0 kB
Buffers:        125388 kB
Cached:          724504 kB
SwapCached:        0 kB
Active:          301532 kB
...
```

4.2.8.2 Un problema imbarazzantemente parallelo risolto in griglia

Determiniamo ora la forma di una funzione $F(x)$ in un range della variabile x (per esempio per trovarne un minimo)⁵.

F viene calcolata con un programma (*computeF*) che accetta in input 3 valori float, cioè gli estremi dell'intervallo di x e la densità dei punti da valutare mentre i risultati vengono restituiti tramite lo **stdout**.⁶

La regione che ci interessa può essere per esempio $[0,1000[$ con punti presi a distanza unitaria.

Dividiamo lo spazio da valutare in due parti: $[0,500[$ e $[500,1000[$ e eseguiamo in metà del tempo il compito. Il JDL sarà:

```
[
    Executable = "computeF";
    Arguments = "0 500 1";

    StdOutput = "stdout";
    StdError = "stderr";

    InputSandbox = {"computeF"};
    OutputSandbox = {"stdout", "stderr"};
]
```

per il primo job (nel secondo basta sostituire la 3a riga con **Arguments** = "500 1000 1";)

Lanciando entrambi i job e unendo i file di output otteniamo il risultato in metà del tempo.

Ovviamente tale suddivisione può essere reiterata in modo da occupare tutte le CPU disponibili e ridurre ulteriormente il tempo di esecuzione.

⁵ quale potrebbe essere la ricerca di percorso minimo per un istanza del problema del *commesso viaggiatore* usando un approccio *brute force*, dove la variabile x rappresenti un percorso tra gli $N!$ disponibili (con N la dimensione del problema)

⁶ per chi volesse eseguire l'esercizio, il comportamento di *computeF* può essere facilmente simulato (per input a valori interi) con uno script del tipo:

```
#!/bin/bash

# Simula il comportamento di una funzione tipo
# computeF. Salvare con nome "computeF".

y=0
for((x=$1;x<$2;x+=3))
do
    # Simula la lentezza dell'algoritmo
    sleep 1
    # valutiamo la funzione y=x*5+1.
    let y=$x*5+1
    echo "X: $x Y: $y"
done
```

4.3 Esercizio 2: Monte Carlo

Ci occuperemo della sottomissione di un programma tipo e dell'automatizzazione del lavoro in griglia.

Non verranno introdotti nuovi comandi di sottomissione ma verrà fatta una panoramica di alcune delle possibilità offerte dal shell scripting per rendere più veloce e efficace il lavoro.

Poichè gli esempi presentati saranno più complicati dei precedenti la comprensione approfondita degli stessi è affidata al lettore che potrà trovare le informazioni sulla BASH o sui comandi che verranno man mano usati nelle referenze; in ogni caso la comprensione dettagliata dei singoli script (abbondantemente commentati) non è un prerequisito alla continuazione del tutorial.

4.3.1 Un semplice esempio di Monte Carlo

Useremo il programma 'area.c' (vedi [Appendice A \[Appendice A\]](#), pagina 60) che calcola l'area di una circonferenza con un semplice metodo Monte Carlo e che legge un intero da un file 'seme.txt' e un numero floating point da 'raggio.txt' e calcola un'approssimazione per difetto dell'area di una circonferenza del raggio specificato; la precisione dell'algoritmo è data da un intero passato a linea di comando con il numero di punti generati.

L'algoritmo genera un insieme di punti in un quadrato che inscrive la circonferenza specificata e salva a intervalli regolari in un file 'output.txt' il numero di punti che sono finiti nella circonferenza, il totale di punti generato e l'area stimata della stessa.

Salviamo il programma in un file 'area.c' e compiliamolo con:

```
gcc -Wall -o area area.c -static7
```

per ottenere un eseguibile chiamato 'area'.

Nella stessa directory creiamo due files: 'seme.txt' e 'raggio.txt' contenenti rispettivamente 123 e 1.0 (per esempio).

Digitiamo ./area 100000 e dovremmo ottenere un file 'output.txt' che conterrà un output simile a questo:

```
7836 9999      3.13471347134713
15723 19999    3.14475723786189
23592 29999    3.14570485682856
31460 39999    3.1460786519663
39377 49999    3.15022300446009
47188 59999    3.14591909865164
55038 69999    3.14507350105001
62954 79999    3.14773934674183
70886 89999    3.15052389470994
78724 99999    3.1489914899149
```

Le conoscenze acquisite finora ci permettono già di eseguire il programma in griglia, il JDL sarà:

```
[
    Executable = "area";
    Arguments  = "100000";
    StdOutput  = "stdout.txt";
```

⁷ l'opzione '-static', aggiunta per completezza, è in questo caso inessenziale non essendoci librerie da includere.

```

StdError    = "stderr.txt";
InputSandbox = {"area","seme.txt","raggio.txt"};
OutputSandbox = {"stdout.txt","stderr.txt","output.txt"};
]

```

che chiameremo ‘area_base.jdl’.

Tralasciando i dettagli sul calcolo dell’errore sulle misure sappiamo che questo compito può essere parallelizzato lanciando su N processori simulazioni più corte di un fattore N e elaborando i risultati sul computer locale, a posteriori (come la maggior parte dei job Monte Carlo si tratta di un problema imbarazzantemente parallelo).

4.3.2 Automatizzazione della sottomissione

Una volta compreso come si possa ottimizzare l’esecuzione di un programma lanciandone diverse copie in parallelo il problema principale diventa eseguire tale compito nella maniera meno impegnativa possibile.

Qui di seguito sono presentate alcune tecniche utili all’esecuzione in griglia di un alto numero di job: dalle possibilità offerte dai comandi di tipo `edg-job-*` all’uso della BASH per complessi script.

4.3.2.1 Salvare i job ID in un file

Nella gestione dei job una difficoltà consiste nella gestione di numerosi ID e quindi dei processi ad essi collegati: le opzioni ‘`--input`’ e ‘`--output`’ (abbreviate rispettivamente con ‘`-i`’ e ‘`-o`’) permettono di superarle.

L’opzione ‘`--output`’ si usa in corrispondenza a `edg-job-submit` e permette di salvare in un file il job ID della sottomissione in corso; il file viene creato se non esiste, altrimenti il nuovo job ID viene messo in coda a quello preesistente.

```
edg-job-submit --vo gridats --output id_file.txt job.jdl
```

L’opzione `--input` si usa con `edg-job-status`, `edg-job-cancel`, `edg-job-get-logging-info` e `edg-job-get-output` per eseguire il comando su alcuni dei job nel file specificato con essa (quali siano viene chiesto all’utente in maniera interattiva).

Per esempio:

```

edg-job-status --input id_file.txt
edg-job-get-output --input id_file.txt
edg-job-cancel --input id_file.txt

```

Per sottomettere diversi file JDL in una directory è così possibile eseguire:

```
# sottomette tutti i job presenti nella directory corrente (ammettendo
# che i file con la loro descrizione abbiano estensione .jdl)
```

```

for i in *.jdl
do
    edg-job-submit --vo gridats --output id_list.txt $i
done

```

per ottenere in ‘`id_list.txt`’ la lista degli ID dei job sottomessi con successo.

Queste opzioni sono sufficienti per job semplici (vedi [Sezione 4.2.8.2 \[Un problema imbarazzantemente parallelo risolto in griglia\]](#), pagina 29) e sono la base degli esempi delle prossime sottosezioni.

Un'altra opzione utile è ‘--noint’ che modifica il comportamento dei comandi che accettano ‘--input’ in modo da non richiedere input all’utente (il comando viene eseguito su tutta la lista di ID).

4.3.2.2 Uso di una template

Una *template* (che può essere tradotta come “modello”) di file JDL è uno dei metodi che permettono di lanciare job con parametri a linea di comando diversi: si crea un file JDL tipo nel quale, ad ogni sottomissione, vengono sostituiti valori diversi usando , per esempio, uno stream editor come sed (Vedi Info file ‘sed’, node ‘sed’).

Tornando al caso di **area** un esempio di template (dove la *keyword* da sostituire è **LUNGHEZZA**) potrebbe essere:

```
[
    Executable = "area";
    Arguments  = "LUNGHEZZA";
    StdOutput  = "stdout.txt";
    StdError   = "stderr.txt";
    InputSandbox = {"area", "seme.txt", "raggio.txt"};
    OutputSandbox = {"stdout.txt", "stderr.txt", "output.txt"};
]
```

che salviamo in ‘template.jdl’.

Lo script che presentiamo qui di seguito (‘template_submission.sh’) gira interamente in locale e si occupa della sottomissione automatica del job: chiede all’utente i parametri della simulazione, crea un file JDL appropriato (usando ‘template.jdl’) e invoca **edg-job-submit** finchè non ha ottenuto gli ID di tutti i job (che vengono salvati nel file ‘id_list.txt’).

Tutti i file generati dallo script hanno un indice che garantisce nomi diversi e quindi permette di risottomettere un job manualmente qualora dovesse abortire durante l’esecuzione.

```
#!/bin/bash

#
# Chiedi all’utente i dati sulla simulazione
#
echo
echo -n "Immetti il raggio: "
read r

echo
echo -n "Quanto deve essere lunga la simulazione? "
read N

echo
echo -n "Su quanti processori vuoi lanciarla? "
read Nproc

let N=$N/$Nproc
```

```
echo
echo -n "Quale desideri sia il primo seme random?"
read seed

echo $r>"raggio.txt"

for((i=0;i<$Nproc;i++))
do
#
# Crea i file con il seme random
#

# Questa copia viene salvata per un'eventuale
# risottomissione.
seedfile=$i.seme.txt
echo "Creazione del seme random in $seedfile."
echo $seed>$seedfile
let seed=$seed+1

#
# Sostituisci i valori appropriati nel jdl.
#

jdlfile=$i.job.jdl
echo "Sto creando il file $jdlfile."
cat template.jdl \
    |sed s/LUNGHEZZA/$N/ > $jdlfile

#
# Questo ciclo ripete la sottomissione finche' non va a buon fine
#

echo "Inizio a sottomettere il job $i"
while true
do
    cp $seedfile seme.txt
    edg-job-submit --noint --vo gridats -o id_list.txt $jdlfile
    if [ $? == 0 ]
    then
        break
    else
        echo "Sottomissione fallita: riprovo in 10 secondi"
        sleep 10
    fi
done
done
```

4.3.2.3 Ancora scripting.

È raro che un programma venga eseguito direttamente (come nei casi precedenti o del primo esercizio (vedi [Sezione 4.2 \[Un primo esempio di sottomissione\]](#), pagina 23): di solito si crea uno script che lo racchiuda così da invocare l'eseguibile in maniera indiretta

Questa tecnica, già accennata in precedenza (vedi [Sezione 4.2.8.2 \[Un problema imbarazzantemente parallelo risolto in griglia\]](#), pagina 29), permette di eseguire diverse operazioni in serie come:

- Eliminare determinati file di output e eventualmente comprimerli prima del ritiro (facilitando la possibilità di usare la Sandbox).
- Usare come input un file tar da scomprimere prima dell'esecuzione del programma (che in aggiunta a quanto detto sulla Sandbox nel punto precedente fa risparmiare sui tempi di sottomissione).
- Eseguire due programmi in modo che il secondo elabori l'output del primo (dimezzando i tempi di sottomissione).
- Eseguire il timing dell'applicazione (con il comando `/usr/bin/time`)).

Il JDL che useremo (e che chiameremo `'wrapper.jdl'`) è una copia dei precedenti:

```
[
    Executable      = "wrapper.sh";
    StdOutput       = "stdout";
    StdError        = "stderr";
    InputSandbox    = {"input.tar.gz", "wrapper.sh"};
    OutputSandbox   = {"output.tar.gz", "stderr", "stdout"};
]
```

`'wrapper.sh'` è il nome dello script che verrà eseguito in griglia e che viene spedito con la Sandbox insieme ad un archivio compresso con i file di input; in output otteniamo un archivio compresso dei risultati.

Non ci sono argomenti per lo script (niente `Arguments`): in questo modo il JDL non deve venire modificato (metteremo l'argomento di `area` in un file).

Anche lo script di sottomissione `'wrapper_submit.sh'` non presenta sorprese (è una versione semplificata di quello precedente): chiede i parametri della simulazione all'utente, genera i files di input, li comprime e sottomette il tutto in griglia.

```
#!/bin/bash
```

```
# Chiedi all'utente i dati sulla simulazione
echo
echo -n "Immetti il raggio: "
read r

echo
echo -n "Quanto deve essere lunga la simulazione? "
read N

echo
echo -n "Su quanti processori vuoi lanciarla? "
read Nproc
let N=$N/$Nproc

# Crea un file nel quale salvare la lunghezza della simulazione
echo $N >sim_length.txt

echo
echo -n "Quale desideri sia il primo seme random?"
read seed

# Crea un file con il raggio
echo $r>"raggio.txt"

for((i=0;i<$Nproc;i++))
do
# Crea i file con il seme random
seedfile=seme.txt
echo "Creazione del seme random in $seedfile."
echo $seed>$seedfile
let seed=$seed+1

# Comprimo i files di input.
tar cvzf input.tar.gz area seme.txt raggio.txt sim_length.txt
```

```

echo "Inizio a sottomettere il job $i"
# Questo ciclo ripete la sottomissione finche' non va a buon fine
while true
do
    edg-job-submit --noint --vo gridats -o id_list.txt wrapper.jdl
    if [ $? == 0 ]
    then
        break
    else
        echo "Sottomissione fallita: riprovo in 10 secondi"
        sleep 10
    fi
done
done

```

La relativa novità è costituita dallo script da eseguire in griglia (`wrapper.sh`) che

- Scomprime l'input.
- *Cambia i permessi del file 'area' rendendolo eseguibile.*
- Esegue il programma con i giusti argomenti (e facendone il timing).
- Rinomina l'output e lo comprime insieme ad altri file interessanti.

```

#!/bin/bash

# Script eseguito sui WN: tutto l'output dello script sara'
# diretto ai file di stdout e stderr specificati nel JDL
# , come
# al solito. L'output del programma verra' rediretto a un file
# specifico.

# Passo 1: scomprimere i dati in input
tar xvzf input.tar.gz

# Passo 2: **cambiare i permessi per l'eseguibile!**
chmod u+x area

# Passo 3: leggere l'argomento da passare a linea di comando
arg=$(cat sim_length.txt)

# Passo 4: eseguire la simulazione usando il comando time per
# raccogliere i dati sulla durata
$(which time) ./area $arg >area_stdout.txt 2>timing.txt

# Passo 5: raccogliere i risultati.
mv output.txt "output_$(cat seme.txt).txt"
tar cvzf output.tar.gz output* area_stdout.txt timing.txt

```

Tutti questi files e l'eseguibile devono essere nella stessa directory.

A questo punto si può procedere alla sottomissione e ritirare i risultati che comprendono (nel file `timing.txt`) una misura della lunghezza della simulazione.

4.3.3 Automatizzare il controllo sui job

Anche se l'output del comando `edg-job-status` con l'opzione `--input` per un numero di job limitato può essere sufficiente, per molti job (nell'ordine delle decine) è necessario filtrare le informazioni.

Lo script che segue (`check_status.sh`) esegue proprio questo compito: legge come primo argomento il nome di un file con una lista di job ID, usa `grep` (Vedi Info file `'grep'`, node `'grep'`) per contare quanti siano in un determinato stato e stampa un riassunto delle informazioni.

```
#!/bin/bash

#
# Controllo che venga fornito il file con gli ID
#
if [ $# -lt 1 ]
then
    echo "Argomenti insufficienti!"
    exit 1
fi

#
# Scarico le informazioni selezionando solo le linee
# che contengono il pattern "Current Status"
#
data=$(edg-job-status --input $1 --noint|grep Current\ Status)

#
# Per ogni possibile stato di un job estraggo le informazioni
# e stampo il risultato.
#
echo "  *** Stato dei jobs ***"
for i in Scheduled Running Done Aborted Cancelled Cleared Ready
do
    echo -e "$i:  \t\t"$(echo $data|tr " " "\n"|grep -c $i)
done
```

eseguendolo con, per es. `./check_status.sh id_list.sh` ottengo così una lista del tipo:

```
> ./check_status.sh id_list.txt
*** Stato dei jobs ***
Scheduled:      5
Running:        0
Done:           1
Aborted:        0
Cancelled:      0
Cleared:        0
```

Ready: 0

Un'altra soluzione, meno elegante ma più semplice e che dimostra le potenzialità delle `coreutils` (Vedi Info file `'info'`, node `'coreutils'`) è:

```
watch "edg-job-status --noint -i id_list.txt \
      |grep Current\ Status|sort|uniq -c"
```

che però non formatta l'output in maniera altrettanto pulita, ma è utile quando si voglia formattare velocemente lo stato dei job senza scrivere uno script.

4.3.4 Automatizzare il ritiro dei risultati

Recuperare le informazioni è spesso la parte più complicata del lavoro in griglia: questo perchè si sovrappone di sovente all'elaborazione dei dati ed il tipo di lavoro da svolgere varia molto da applicazione ad applicazione.

Per il ritiro vero e proprio il comando `edg-job-get-output` con le opzioni `'--input'`, `'--noint'` e `'--dir'` è sempre sufficiente⁸: sono l'estrazione delle informazioni utili e l'elaborazione a essere difficili da automatizzare.

Lo script che segue (che chiamiamo `'get_results.sh'`) accetta in input un file di ID generato con lo script `'wrapper_submit.sh'` (vedi [Sezione 4.3.2.3 \[Wrapping dei job\], pagina 34](#)) e si occupa di:

1. Richiedere i dati dei job "Done".
2. Estrarre i files di output dalle directory.
3. Raccogliere le informazioni e creare un file `'results.txt'` nel quale scrivere 3 colonne: il numero di punti generati in totale, il numero di punti entro la circonferenza sommati su tutti i job (dei quali l'output è già stato scaricato) e una stima di π .

```
#!/bin/bash

#
# Controllo che venga fornito il file con gli ID
#
if [ $# -lt 1 ]
then
    echo "Argomenti insufficienti!"
    exit 1
fi

echo
echo "Scarico i risultati..."
edg-job-get-output --input $1 --noint --dir .
```

⁸ e può essere eseguito anche su liste di ID che contengono job non ancora terminati o già scaricati senza alcuna complicazione

```

#
# Per ogni directory del tipo "nomeutente_*"
# estraggo i file del tipo output_ (i nomi sono unici)
#
echo
echo "Inizio l'estrazione dei file output_*"
for i in $(whoami)_*
do
    echo "Processing directory $i"
    tar xvzf $i/output.tar.gz 'output_*' >/dev/null
done

echo
echo "Inizio elaborazione dei dati..."
echo "(potrebbe essere necessario qualche minuto)"

#
# In un file temporaneo salvo la colonna con il numero
# di punti generati in totale sul singolo processo e li
# multiplico per il numero di processi
#
echo
echo "Creazione dell'indice"
tmpfile=$(mktemp)
echo $tmpfile
nfiles=$(ls output_*.txt |grep -c output_)
cat $(ls output_*.txt|head -n 1)          \
    |gawk '{print $2"*'$nfiles'}' \
    |bc >$tmpfile

#
# In un altro file temporaneo salvo tutte le colonne con
# i punti che sono finiti in una circonferenza.
#
echo
echo "Salvataggio delle colonne coi dati"
tmpfile2=$(mktemp)
nfiles=0
for i in output_*.txt
do
    let nfiles=$nfiles+1
    cat $i|gawk '{print $1}'|paste $tmpfile2 - >results.txt
    mv results.txt $tmpfile2
done

```

```

#
# Ora sommo tutte le colonne con i dati per poi aggiungerle
# alla colonna con i punti generati in totale, nel file results.txt
#
echo
echo "Creazione del gran totale"
cat $tmpfile2|gawk '{
    stringa="0";
    for(i=1;i<=NF;i++)
        stringa=stringa+"$i;
    print stringa
}'|bc -l|paste $tmpfile - > results.txt

#
# metto in coda la stima di PI
#
cp results.txt $tmpfile
cp results.txt $tmpfile2

# Aggiungo un header (che venga letto come commento da gnuplot
# per informazioni, man gnuplot)

echo -e "# Tot.    Int.        PI(stima)" >results.txt

cat $tmpfile |gawk '{
    print "4*"$2/"$1
}'|bc -l |paste $tmpfile - >> results.txt

#
# Rimuovo i file temporanei che altrimenti occuperebbero
# spazio inutile sull'hard disk
#
echo
echo "Rimozione dei files temporanei..."
rm -f $tmpfile2 $tmpfile

```

Lo script è piuttosto complesso e fa uso di comandi come `bc` (*man bc*) che serve per eseguire calcoli in multiprecisione o con virgola mobile (e in questo caso viene usato per leggere da **stdin** le operazioni da eseguire), `gawk` che è un'utility complessa per l'elaborazione di files di testo Vedi Info file '`gawk`', node '`Gawk`' e `mktemp` (*man mktemp*); d'altro canto il tempo necessario per scriverlo e correggerlo è stato di circa 1 ora (contro le diverse ore, se non giorni, necessari per mettere a punto un programma equivalente in **C**, **FORTRAN** o simili).

L'elaborazione automatica dei dati risulta estremamente utile per il lavoro in griglia che altrimenti sarebbe impratico: il prezzo che si paga usando gli script è in termini di prestazioni: spetta all'utente decidere tra programmi veri e propri (difficili da codificare ma veloci) e scripts (facili da comporre ma estremamente lenti) basandosi sulle caratteristiche del problema.

4.3.5 Linee guida per la completa automatizzazione

Quanto visto finora semplifica certamente il lavoro in griglia, ma è solo una parte di ciò che si può fare per rendere la sottomissione un'operazione user-friendly.

Senza entrare nel dettaglio (che non è negli obiettivi di questo tutorial), un sistema completo di sottomissione dovrebbe comprendere, oltre agli script già visti, anche uno capace di controllare lo stato della simulazione e risottomettere dei job in caso di fallimento.

Tale compito, in apparenza triviale, viene complicato da una serie di dettagli come il fatto che tutti i dati usati per la sottomissione⁹ devono essere salvati e collegati al job ID al quale appartengono e che lo script deve modificare la lista degli ID (o crearne una nuova) per rimuovere o inserire job ID.

In conclusione, un set di utility per la sottomissione di un programma in griglia può dirsi completo se ci sono:

- Una procedura di sottomissione che accetta dall'utente i parametri dell'applicazione in maniera opportuna e salva gli ID dei job (ed eventualmente altri dati).
- Uno script per il controllo dello stato della simulazione (che in linea di principio è sempre uguale da programma a programma).
- Una procedura automatica di risottomissione per controllare che tutti i job vadano a buon fine, che controlla lo stato dei job e risottomette *con i parametri opportuni* quelli falliti.
- Uno o più script per il ritiro dei dati, che nei casi più semplici si riducono a `edg-job-get-output` e in quelli più complessi a veri e propri programmi che leggono i files risultanti e ne elaborano il contenuto.

che in un'implementazione ideale agiscono sinergicamente (per esempio chiamati da un ulteriore script che li coordina) e con una minima interazione da parte dell'utente.

4.4 Esercizio 3: Monte Carlo in MPI

Useremo una versione **MPI** enabled del programma **area** per mostrare la sottomissione di programmi compilati con le librerie MPICH: questo approccio permette di usare quasi senza modifiche programmi che sono stati progettati per cluster di computer.

4.4.1 Un esempio MPI

Il programma è una versione modificata di quello dell'esercizio precedente, **areaMPI** (vedi [Sezione A.2 \[areaMPI.c\], pagina 61](#)) esegue lo stesso compito del predecessore ma in più si sincronizza con i processi del suo pool per stampare il risultato del conto mediato su tutti i processori.

⁹ quale per esempio un eventuale seme random

I parametri accettati sono gli stessi di `'area.c'` anche se ora vengono letti da un file `'params.txt'`, su righe diverse e nell'ordine: seme, interazioni e raggio.

Il numero di steps viene diviso per il numero di CPU richieste e il seme random incrementato con l'indice del processore sul quale si stanno facendo i conti poi, ogni ogni 10000 passi vi è la sincronizzazione e la stampa su file dei risultati.

4.4.2 Librerie utilizzate

Per poter usufruire del supporto **MPI** occorre avere una versione recente delle librerie MPICH (reperibili sul sito <http://www-unix.mcs.anl.gov/mpi/mpich/index.htm>) nella versione **p4** (per l'esecuzione su *egrid-ce-01.egrid.it*).

4.4.3 JDL per jobs MPI

I job **MPI** necessitano di maggior informazioni da parte dell'utente di quelli seriali poichè è necessaria la condivisione e sincronizzazione delle risorse su WN multipli con la conseguenza che il gestore di code del CE (vedi Sezione 4.1.1 [Gestione dei jobs], pagina 22) determina il modo con il quale il programma dovrà essere sottomesso.

Attualmente **GRID@Trieste** ospita un cluster con sistema di code *lcpbs* che ha la caratteristica di non condividere le *home directory* sui vari nodi: come conseguenza di ciò l'utente deve copiare manualmente i dati necessari al programma (compreso l'eseguibile!) su ogni WN tramite uno script¹⁰.

Tutti i job **MPI** devono essere identificati come tali nel file JDL e specificare il numero di processori richiesti¹¹ in questo modo:

```
JobType = "MPICH";
NodeNumber = numero processori;
```

Inoltre anche se attualmente non sarebbe necessario (è presente solo un cluster **MPI** in **GRID@Trieste** che verrebbe scelto in ogni caso) la griglia può essere informata riguardo il tipo di architettura che stiamo utilizzando:

```
Requirements = other.GlueCEInfoLRMSType == "PBS" &&
Member("p4", other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Con queste aggiunte, il JDL (`'areaMPI.jdl'`) sarà:

```
[
Executable   = "areaMPI.sh";
Arguments    = "areaMPI 2";
JobType      = "MPICH";
NodeNumber   = 2;
StdOutput    = "stdout.txt";
StdError     = "stderr.txt";
InputSandbox = {"areaMPI", "areaMPI.sh", "params.txt"};
OutputSandbox = {"stderr.txt", "stdout.txt", "output.txt"};
Requirements = other.GlueCEInfoLRMSType == "PBS" &&
Member("p4", other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

¹⁰ altri sistemi di code, come *pbs* si comportano diversamente e permettono quindi di sottomettere in maniera diretta senza operazioni supplementari)

¹¹ che in questo caso deve essere pari per questioni legate all'architettura SMP

]

dove 'areaMPI.sh' è lo script che copierà i dati sui vari WN e invocherà l'eseguibile, cioè:

```
#!/bin/sh
#
# the first parameter is the binary to be executed
#

EXE=$1

# this parameter is the number of CPU's to be reserved for
# parallel execution
CPU_NEEDED=$2

# prints the name of the master node
echo "Running on: $HOSTNAME"

echo "*****"
if [ -f "$PWD/.BrokerInfo" ] ; then
    TEST_LSF='edg-brokerinfo getCE | cut -d/ -f2 | grep lsf'
else
    TEST_LSF='ps -ef | grep sbatchd | grep -v grep'
fi

if [ "x$TEST_LSF" = "x" ] ; then
    # prints the name of the file containing the nodes allocated for
    # parallel execution
    echo "PBS Nodefile: $PBS_NODEFILE"
    # print the names of the nodes allocated for parallel execution
    cat $PBS_NODEFILE
    echo "*****"
    HOST_NODEFILE=$PBS_NODEFILE
else
    # print the names of the nodes allocated for parallel execution
    echo "LSF Hosts: $LSB_HOSTS"
    # loops over the nodes allocated for parallel execution
    HOST_NODEFILE='pwd'/lsf_nodefile.$$
    for host in ${LSB_HOSTS}
    do
        echo $host >> ${HOST_NODEFILE}
    done
fi

echo "*****"
# prints the working directory on the master node
echo "Current dir: $PWD"
```

```

echo "*****"

for i in `cat $HOST_NODEFILE` ; do
    echo "Mirroring via SSH to $i"
    # creates the working directories on all the nodes allocated for
    # parallel execution
    ssh $i mkdir -p `pwd`
    # copies the needed files on all the nodes allocated for parallel
    # execution
    /usr/bin/scp -rp ./* $i:`pwd`
    # checks that all files are present on all the nodes allocated
    # for parallel execution
    echo `pwd`
    ssh $i ls `pwd`
    # sets the permissions of the files
    ssh $i chmod 755 `pwd`/$EXE
    ssh $i ls -alR `pwd`
    echo "oooooooooooooooooooo"
done

# execute the parallel job with mpirun

echo "*****"
echo "Executing $EXE"
chmod 755 $EXE
ls -l
#
# Mpi execution
#
mpirun -np $CPU_NEEDED \
    -machinefile $HOST_NODEFILE `pwd`/$EXE > executable.out
echo "*****"

```

Anche se questo script (che chiameremo `lcgpbs_MPI.sh`) è complicato, non necessita mai di essere modificato visto che è tutto quanto occorre per la sottomissione in un sistema di code **lcgpbs** di qualunque job **MPI**.

Il primo parametro che viene passato è il nome dell'eseguibile mentre il secondo è il numero di computer richiesti (che deve coincidere con quelli in `NodeNumber`): a questo punto lo script cerca i nomi dei computer che sono stati assegnati all'utente, vi crea una directory uguale a quella locale e fa partire il programma.

Con queste considerazioni, la sottomissione di un job **MPI** diventa del tutto analoga a quella di un job singolo.

4.5 Esercizio 4: caricare i dati direttamente dalla griglia

Il meccanismo della Sandbox è comodo per l'upload e il download di piccole quantità di dati ma diventa impratico per file grandi, specialmente quando questi debbano essere rispediti a ogni sottomissione.

Il salvataggio dei dati direttamente in uno *Storage element* (o SE), in griglia, ovvia a questo problema.

Poichè i concetti illustrati in questo esercizio si sovrappongono parzialmente al capitolo che tratta dei dati in maniera specifica (vedi [Capitolo 5 \[Operazioni sui dati\]](#), pagina 49) la teoria dello spostamento e della gestione degli stessi verrà rimandata di conseguenza: in questa sezione verranno mostrate solo le opzioni che influenzano la sottomissione e il ritiro dei job nei file JDL.

4.5.1 Input di un job dalla griglia

Il caricamento dei files in griglia non viene gestito automaticamente dal sistema: l'utente deve utilizzare i comandi di tipo `lcg-*` per copiare i dati da un SE al WN vedi [Sezione 5.2.4 \[lcg-cr\]](#), pagina 52.

Per rendere più veloce il trasferimento è disponibile un'opzione nei file JDL con la quale specificare un file in griglia che verrà poi scaricato (manualmente dall'utente) sul WN; l'azione di questa opzione è di far scegliere un WN prossimo ad uno Storage Element che contenga una replica del file (vedi [Sezione 5.3.2 \[Le repliche\]](#), pagina 57).

Con le opzioni che seguono la griglia sottometterà un job solo sui WN che sono sullo stesso CE (nella stessa rete locale) di un SE con una replica di `'lfn:/grid/gridats/crossi/dati.dat'` (vedi [Capitolo 5 \[Operazioni sui dati\]](#), pagina 49).

```
InputData = "lfn:/grid/gridats/crossi/dati.dat";
DataAccessProtocol = "gsiftp";
```

L'opzione `DataAccessProtocol = "gsiftp"`; va specificata una sola volta

Una volta in esecuzione sarà responsabilità dello script caricare i dati sul WN (con il comando `lcg-cp`); a questo proposito è importante notare che nulla vieta di caricare i dati dalla griglia senza specificarlo nel file JDL: l'opzione appena vista serve solo per avvicinare i job all'SE che li contiene.

Questo metodo è conveniente solo se ci sono numerose repliche su diversi SE, altrimenti riduce il numero di WN accessibili per un determinato job (ammesso che ve ne siano!).

4.5.2 Output di un job dalla griglia

A differenza di quanto accade per la lettura, la scrittura dei dati al termine del job può essere gestita in maniera automatica dal sistema con l'opzione `OutputData`, per esempio:

```
OutputData={
[
  OutputFile="risultati.dat";
  StorageElement="sfn://egrid-ce-01.egrid.it/(//)
    storage/gridats/crossi/risultati.dat";
  LogicalFileName="lfn:/grid/gridats/crossi/risultati.dat";
]
};
```

ordina alla griglia di salvare alla fine del job il file locale (rispetto al WN) ‘risultati.dat’ come ‘sfn://egrid-10.egrid.it/tmp/risultati3.txt’, con nome logico ‘lfn:/grid/gridats/aterpin/risultati3.txt’.

È inoltre possibile specificare solo il nome dell’*host* per `StorageElement` e lasciare che la destinazione del file sia automatica: per farlo basta omettere il prefisso ‘sfn://’ (per esempio: ‘StorageElement = "egrid-ce-01.egrid.it"’).

È comunque possibile salvare i dati in maniera manuale: quanto detto nel capitolo sulla gestione dei files vale per quanto concerne i WN (vedi [Capitolo 5 \[Operazioni sui dati\]](#), [pagina 49](#)).

4.5.3 Un semplice esempio

Senza entrare nei dettagli sui comandi vediamo ora come fare un semplice test delle funzionalità dei dati durante la sottomissione.

Iniziamo controllando quali siano gli SE a nostra disposizione (vedi [Sezione 5.2.2 \[egrid-infosites\]](#), [pagina 50](#)) e di quanto spazio dispongano:

```
bash$ lcg-infosites --vo gridats se ce
```

e vediamo che ‘baciuco.hpc.sissa.it’ ha sia spazio che CPU disponibili:

```
*****
These are the related data for gridats: (in terms of queues and CPUs)
*****

#CPU      Free      Total Jobs Running Waiting ComputingElement
-----
    28       28         0         0         0  baciuco.hpc.sissa.it:2119/  (//)
                                           jobmanager-pbs-gridats
```

```
*****
These are the related data for gridats: (in terms of SE)
*****
```

Avail Space(Kb)	Used Space(Kb)	Type	SEs
5772196	3796104	disk	baciuco.hpc.sissa.it

Creiamo un file sul filesystem locale così da avere qualcosa da copiare in griglia:

```
bash$ echo 'Grid@TS Rules!' > locale
```

e poi copiamolo sull’SE scelto:

```
bash$ lcg-cr --vo gridats \
      -l lfn:/grid/gridats/crossi/remoto \
      -d baciuco.hpc.sissa.it \
```

```
file:'pwd'/locale \
```

che tornerà una stringa del tipo:

```
guid:8f48d0e9-b103-46a5-a958-38be0c52c882
```

Ora¹² dovremmo essere in grado di vedere il file copiato con:

```
bash$ lfc-ls -l /grid/gridats/crossi/remoto
```

ottenendo:

```
-rwxrwxr-x 1 60115 60115 15 Sep 21 15:54 /grid/gridats/crossi/remoto
```

e di controllare che esista una copia fisica del file, usando

```
bash$ lcg-lr lfn:/grid/gridats/crossi/remoto
```

che tornerà qualcosa come:

```
sfn://baciuco.hpc.sissa.it/storage/gridats/generated/2005-09-29/file9807aac5-5202-4846-9176-9af2e93d80c3
```

A questo punto possiamo caricare il file salvato in griglia da uno dei WN, modificarlo e ricopiarlo (con nome diverso) di nuovo nella nostra directory sull' SE.

Usando le nozioni delle sezioni precedenti possiamo creare un file jdl come il seguente ('file_operations.jdl'):

```
[
  Executable= "file_operations.sh";

  StdOutput = "stdout";
  StdError  = "stderr";

  InputSandbox  = {"file_operations.sh"};
  OutputSandbox = {"stderr", "stdout"};

  DataAccessProtocol="gsiftp";
  InputData = {"lfn:/grid/gridats/crossi/remoto"};

  OutputData={
    [
      OutputFile="modificato";
      StorageElement="baciuco.hpc.sissa.it";
      LogicalFileName="lfn:/grid/gridats/crossi/modificato";
    ]
  };
]
```

mentre lo script che verrà eseguito ('file_operations.sh') sarà:

```
#!/bin/bash

# Controlla il contenuto della directory a inizio job.
ls -l

# Scarichiamo il file dall' SE
```

¹² se avremo avuto cura di sostituire crossi con il nostro *username* di griglia

```
lcg-cp --vo gridats lfn:/grid/gridats/crossi/remoto file:'pwd'/remoto
```

```
# Verifichiamo che sia stato effettivamente scaricato
# e stampiamone il contenuto
ls -l remoto
cat remoto
```

```
# Creiamo un altro file (da salvare in griglia)
cat remoto \
  |sed s/Rules/Rules\ a\ lot/ \
  > modificato
```

```
# Non occorre copiare manualmente il file sull'SE!
# La griglia se ne occuperà automaticamente!
#nop
```

Dopo aver sottomesso quest'esempio, verifichiamo che sulla griglia sia presente 'modificato':

```
bash$ lfc-ls -l /grid/gridats/crossi/modificato
```

e scarichiamolo sul filesystem locale:

```
bash$ lcg-cp --vo gridats \
  lfn:/grid/gridats/crossi/modificato \
  file:'pwd'/mangled-local_file.txt
```

Con un 'cat modificato' dovremmo otterremo:

```
Grid@TS Rules a lot!
```

che dimostra che il file è stato modificato in griglia, come desiderato.

5 Gestire i file in griglia

La gestione dei dati in griglia avviene su due livelli logici distinti,

- Livello Fisico: Storage Element
- Livello Logico: Catalogo

Gli *Storage Element* (SE) sono i computer che memorizzano i file. L'accesso ad uno storage element avviene attraverso un protocollo simile a ftp ma che richiede un certificato proxy valido, cfr. [Sezione 3.4 \[Autenticazione\]](#), [pagina 12](#).

Ad ogni file fisico contenuto su uno storage element viene associato un nome logico (*Logical File Name*, o *LFN*). Tali corrispondenze sono conservate in un *Catalogo* che fornisce l'illusione di un unico filesystem **UNIX**. Il catalogo non è altro che un database MySQL che contiene i *metadati* dei file, ovvero informazioni quali il nome, la dimensione, i permessi e le date di accesso/modifica del file stesso.

Per poter accedere ad un file è sufficiente conoscerne il nome logico o la locazione fisica.

5.1 Il catalogo LFC - Introduzione

Ad ogni file in griglia viene associato un identificativo unico, chiamato *Grid Unique Identifier* o *GUID*. Un GUID è della forma:

```
guid:4fb301ed-8f6d-4f00-8faa-d68873571128
```

Ad ogni GUID, ovvero ad ogni file viene associato uno o più nomi logici di facile memorizzazione (LFN), perfettamente analoghi ai percorsi dei file in **UNIX**. Tale percorso è nella forma:

```
lfn:/grid/VO_NAME/mia_directory/filetest
```

Ogni file risiede su uno o più Storage Element, perciò ad ogni GUID (e quindi ad ogni LFN) viene associato almeno un *Site URL* (*SURL*) che specifica la locazione fisica del file. Un SURL può essere della forma

```
sfn://egrid-3.gridats.it/storage/grid/gridats/crossi/MioFile.test
```

o

```
srm://StormServer/Path del file
```

e la differenza tra le due forme è esclusivamente tecnica: dipende da come i vari servizi comunicano tra di loro e non ha perciò nessun interesse per l'utente finale. L'utente con tutta probabilità potrà operare esclusivamente usando i nomi logici.

La copia di un file in griglia avviene come due azioni distinte: l'upload del file su un SE e la registrazione sul catalogo con conseguente associazione tra un SURL ed una coppia LFN+GUID. Questa distinzione passerà del tutto sotto silenzio per la maggior parte delle operazioni che l'utente dovrà eseguire.

In alcuni casi può essere utile conservare più copie di un medesimo file su più SE. Tali copie si chiamano *repliche* e possono essere create con il comando `lcg-rep`. Una replica può essere creata dopo che il file è stato correttamente memorizzato su un SE e registrato nel catalogo. A quel punto è possibile creare una copia del file stesso su un altro SE. LFN e GUID resteranno gli stessi, ma verrà associato al file un SURL aggiuntivo che specificherà la posizione sul nuovo SE.

I comandi che operano esclusivamente sul catalogo sono del tipo `lfc-*` e identificano il catalogo da usare grazie alla variabile d'ambiente `LFC_HOST`. Quelli che operano anche sui file veri e propri, e che si occupano del download e dell'upload di file sui vari SE sono del tipo `lcg-*` e richiedono *tutti* l'opzione `--vo` seguita dal nome della Virtual Organization.

Per poter sapere quali sono gli SE disponibili e per avere informazioni sulla struttura della griglia si vedano le sezioni [Sezione 5.2.2 \[Reperire informazioni sulla struttura della griglia\]](#), pagina 50 e [Sezione 5.2.3 \[Reperire informazioni sulla struttura della griglia \(lcg-infosites\)\]](#), pagina 52.

Le operazioni principali che si possono fare con i file in griglia sono l'upload di file in griglia, il download, la rimozione, cambiare il nome, e creare più nomi logici per accedere al medesimo file. La sezione [Sezione 5.2 \[Comandi di base\]](#), pagina 50 spiega i comandi necessari per eseguire questi e più specializzati compiti, mentre comandi più complicati (tra cui quelli necessari a creare le repliche) sono spiegati nella sezione [Sezione 5.3 \[Argomenti avanzati\]](#), pagina 56.

5.2 Comandi di uso frequente

5.2.1 Ricerca dei file nel catalogo

`lfc-ls` mostra il contenuto di una particolare directory del catalogo. Ammette una unica opzione e necessita di un parametro, che sarà il LFN meno la prima parte che identifica il tipo di URL (ovvero `'lfn:.'`). Quindi per vedere il contenuto di `'lfn://grid/gridats'` il comando da dare sarà:

```
bash$ lfc-ls /grid/gridats
Datadir
data
alias
```

L'opzione `'-l'` funziona esattamente come la corrispondente opzione del comando `ls`, e mostra ulteriori informazioni sui file, permettendo così di identificare directory, alias e file:

```
bash$ lfc-ls -l /grid/gridats
drwxr-xr-x  1 506  504      0 Aug 17 10:30 Datadir
-rwxrwxr-x  1 506  504 1645 Aug 10 17:57 data
lrwxrwxrwx  1 506  504      0 Aug 10 18:43 alias -> /grid/gridats/data
```

Gli alias, cfr. [Sezione 5.2.9 \[Creare più nomi logici per il medesimo file\]](#), pagina 55, vengono visti come link simbolici.

5.2.2 Reperire informazioni sulla struttura della griglia (egrid-infosites)

Il comando `egrid-infosites` permette di visualizzare utili informazioni sulla griglia. In particolare permette di recuperare informazioni su:

- CE (*Computing Element*)
- SE (*Storage Element*)
- i tag supportati dai vari CE
- L'host che gestisce il catalogo
- l'SE più vicino ad ogni CE

- I system administrator dei vari CE

La sintassi è la seguente:

```
egrid-infosites --vo VO (ce|close|lfc|lrc|rmc|se|tags|all)
```

‘--vo VO’ Specifica la virtual organization da usare

‘(ce|close|lfc|lrc|rmc|se|tags|all)’

Il primo argomento specifica quali informazioni si vogliono visualizzare:

‘ce’	Mostra informazioni sui CE
‘close se’	Mostra gli SE più vicini ai vari CE
‘lfc’	Mostra l’hostname del computer che conserva il catalogo attualmente utilizzato
‘lrc’	Mostra l’hostname del computer che conserva parte del catalogo LRC (obsoleto e non utilizzato)
‘rmc’	Mostra l’hostname del computer che conserva parte del catalogo RMC (obsoleto e non utilizzato)
‘se’	Mostra informazioni sugli SE
‘tags’	Mostra informazioni aggiuntive relative alle code dei CE (in particolare specificano alcune caratteristiche peculiari del CE, ad esempio se supportano MPICH o meno)
‘all’	Mostra <i>tutte</i> le informazioni sopramenzionate

Ad esempio, se si volesse usare il comando `lcg-cr` o se si volesse creare una replica ad un file si dovrebbe conoscere almeno l’hostname di un SE. Per reperire la lista di SE validi basterà:

```
bash$ egrid-infosites --vo gridats se
```

SE	Protocols	Avail. Space	Used Space
baciuco.hpc.sissa.it	rfio,gsiftp	6477184	3091116
beryllium.hpc.sissa.it	gsiftp	32493592	2185720
egrid-10.egrid.it	gsiftp	32798100	2781360
egrid-3.egrid.it	rfio,gsiftp	35337316	8579784
egrid-ce-01.egrid.it	rfio,gsiftp	14681376	32884

mentre se si volesse conoscere l’elenco dei CE disponibili il comando da dare sarebbe:

```
bash$ egrid-infosites --vo gridats ce
```

CE	R W CPUs	MaxCPU	MaxTime
baciuco.hpc.sissa.it:2119/jobmanager-pbs-egrid	28	2880	4320

```

|egrid-3.egrid.it:2119/jobmanager-lcgpbs-egrid      | | |2   |2880  |4320  |
|egrid-ce-01.egrid.it:2119/jobmanager-lcgpbs-egrid  | | |8   |2880  |4320  |
(...)

```

5.2.3 Reperire informazioni sulla struttura della griglia (lcg-infosites)

Assolutamente analogo a `egrid-infosites`, `lcg-infosites` ha una sintassi lievemente diversa.

```
lcg-infosites --vo VO (ce|closeSE|se|tag|all)
```

‘--vo VO’ Specifica la virtual organization da usare

‘(ce|closeSE|se|tag|all)’

Il primo argomento specifica quali informazioni si vogliono visualizzare:

- ‘ce’ Mostra informazioni sui CE. Tra le informazioni visualizzate citiamo l’hostname comprensivo di porta di accesso e tipo di servizio (CE), il numero di job in esecuzione (**R**), il numero di job che attendono di essere eseguiti (**W**), il numero di CPU che compongono il CE.
- ‘closeSE’ Mostra gli SE più vicini ai vari CE
- ‘se’ Mostra informazioni sugli SE
- ‘tag’ Mostra informazioni aggiuntive relative alle code dei CE (nota: attualmente non funziona)
- ‘all’ Mostra *tutte* le informazioni sopramenzionate

ATTENZIONE: il comando `lcg-infosites` in alcuni casi fornisce informazioni parziali, quindi in caso di dubbi relativamente alle informazioni ricevute usate anche il comando `egrid-infosites`.

5.2.4 Copiare file locali sul filesystem di griglia

Per copiare un file su una griglia è necessario conoscere almeno l’hostname di un SE, quindi si legga prima la sezione [Sezione 5.2.2 \[Reperire informazioni sulla struttura della griglia\]](#), [pagina 50](#). La sintassi di base di `lcg-cr` è la seguente:

```
lcg-cr -d destination --vo VO [-l lfn] src_file
```

‘-d destination’

Si può specificare un URL o semplicemente l’hostname di un SE. Per avere l’elenco degli SE disponibili usare il comando `egrid-infosites`

‘-l lfn’ Questa opzione non è necessaria. Se specificata il file verrà registrato con il nome logico scelto, altrimenti sarà assegnato un nome logico casuale

‘--vo VO’ Specifica la virtual organization da usare

‘src_file’

Specifica il file sorgente. Deve essere specificato anche il protocollo, che può essere ‘file:’ o ‘gsiftp:’

Il comando `lcg-cr` si preoccupa di copiare il file sull'SE specificato e di registrarlo con un nome logico (scelto dall'utente con l'opzione `-l` o generato automaticamente).

Ovviamente non è detto che si conosca l'hostname di un SE valido, quindi potrebbe essere necessario usare il comando `egrid-infosites`:

```
bash$ egrid-infosites --vo gridats se
+-----+-----+-----+-----+
|SE              |Protocols |Avail. Space|Used Space|
+-----+-----+-----+-----+
|baciuco.hpc.sissa.it |rfio,gsiftp|6477184     |3091116   |
|beryllium.hpc.sissa.it|gsiftp     |32493592    |2185720   |
|egrid-10.egrid.it    |gsiftp     |32798100    |2781360   |
|egrid-3.egrid.it     |rfio,gsiftp|35337316    |8579784   |
|egrid-ce-01.egrid.it |rfio,gsiftp|14681376    |32884     |
```

a questo punto potremo scegliere un SE (ad esempio `'egrid-ce-01.egrid.it'`) e dare il finalmente il comando `lcg-cr`. In caso di successo restituirà come output il GUID del file:

```
bash$ lcg-cr --vo gridats -l lfn:/grid/gridats/test.txt \
                                     -d egrid-ce-01.egrid.it \
                                     file:/home/crossi/test.txt
guid:298c666e-9792-4a69-9c5a-6594d755b3eb
```

Adesso la verifica dell'esistenza del file può essere fatta con un semplice

```
bash$ lfc-ls -l /grid/gridats/test.txt
-rwxrwxr-x 1 60125 60125 0 Aug 22 14:52 /grid/gridats/test.txt
```

5.2.5 Prelevare file dalla griglia sul computer locale

Per prelevare un file dalla griglia è sufficiente conoscerne il nome logico. È possibile però utilizzare anche il GUID o il SURL, se se ne è a conoscenza. La sintassi di base del comando è:

```
lcg-cp --vo VO grid_file local_file
```

`'--vo VO'` Specifica la virtual organization da usare

`'grid_file'`

Specifica il file da prelevare. Deve essere specificato anche il protocollo, che può essere `'lfn:'`, `'guid:'`, `'sfn:'` o `'srm:'`

`'local_file'`

Specifica la URL locale in cui copiare il file, quindi il percorso assoluto del file da creare preceduto da `'file:'`.

Esempio:

```
bash$ lcg-cp --vo gridats lfn:/grid/gridats/pippo file:$(pwd)/pippo
```

5.2.6 Rimozione di file

Per eliminare sia il file fisico che la relativa entry nel catalogo si usa il comando `lcg-del`. La sintassi di base è anche in questo caso semplice:

```
lcg-del --vo (-a|-s SE) VO grid_file
```

`--vo VO` Specifica la virtual organization da usare

`(-a|-s SE)`

Specifica se si vuole eliminare solo una delle repliche o tutte. Si deve selezionare perciò almeno una delle due opzioni:

`-a` elimina tutte le repliche del file

`-b SE` SE deve essere uno degli storage element su cui il file risiede. In questo caso verrà eliminata solamente la replica memorizzata nello Storage Element specificato

`grid_file`

Specifica il file da prelevare. Deve essere specificato anche il protocollo, che può essere `lfn:`, `guid:`, `sfn:` o `srm:`

Supponiamo quindi di avere:

```
bash$ lfc-ls -l /grid/gridats/crossi
drwxr-xr-x  1 506  504      0 Aug 17 10:30 Datadir
-rwxrwxr-x  1 506  504 1645 Aug 10 17:57 file1
-rwxrwxr-x  1 506  504 1645 Aug 11 17:57 delme
```

e di voler cancellare `delme` su tutti i possibili SE sui quali risiede, sarà sufficiente dare:

```
bash$ lcg-del -a --vo gridats lfn:/grid/gridats/crossi/delme
bash$ lfc-ls -l /grid/gridats/crossi
drwxr-xr-x  1 506  504      0 Aug 17 10:30 Datadir
-rwxrwxr-x  1 506  504 1645 Aug 10 17:57 file1
```

5.2.7 Rinomina di file

È possibile rinominare file o directory usando il comando `lfc-rename`. La sintassi è:

```
lfc-rename old_path new_path
```

dove `old_path` e `new_path` sono nomi logici e non si deve specificare il protocollo (ovvero non si deve premettere `lfn:`). Ad esempio:

```
bash$ lcg-rename /grid/gridats/crossi/pippo.txt /grid/gridats/crossi/pluto.txt
```

La rinomina agisce solamente a livello sul LFN, quindi è un'azione che influisce esclusivamente sul catalogo.

5.2.8 Creazione e rimozione di directory

Le directory non risiedono fisicamente su nessun SE ma sono un'astrazione del catalogo, perciò la loro creazione serve esclusivamente per organizzare in maniera più logica i propri file.

Per creare una directory si usa il comando `lfc-mkdir`. La sintassi:

```
lfc-mkdir [-m numeric_mode] [-p] path
```

dove:

`-m numeric_mode`

specifica i permessi in perfetta analogia ad un filesystem **UNIX**. `numeric_mode` deve essere un valore numerico; il valore di default è 0777. Questa opzione non è necessaria

‘-p’ In caso sia necessario crea anche le directory superiori, se non esistono (e.g. se si vuole creare ‘/uno/duetre’ ma ‘/uno/duet’ non esiste allora verrà creata).

‘path’ Specifica il nome logico della directory (non si usa ‘lfn:’)

Per rimuovere directory (che non risiedono fisicamente su nessun filesystem) il comando da usare è `lfc-rm`, e la sintassi è semplicemente:

```
lfc-rm -r path
```

dove *path* è il nome logico della directory (non si usa ‘lfn:’)

È possibile rinominare directory nella stessa maniera in cui si rinominano file usando il comando `lfc-rename`. Vedere [Sezione 5.2.7 \[Rinomina di file\]](#), pagina 54.

5.2.9 Creare più nomi logici per il medesimo file

È possibile associare al medesimo file più nomi logici. I nomi logici successivi al primo vengono chiamati *alias*, e per crearli esistono due comandi assolutamente analoghi: `lcg-aa` e `lfc-ln`.

La sintassi di `lfc-ln` è elementare:

```
lfc-ln -s original_name (new_alias|directory)
```

‘original_name’

Il nome logico del file del quale si vuole creare un alias (non si usa ‘lfn:’).

‘new_alias|directory’

In perfetta analogia con il comando UNIX ‘`ln -s`’ si può creare un alias specificando il nome completo oppure specificando semplicemente il nome della directory all’interno della quale vogliamo creare l’alias; in questo caso verrà creato un alias con il medesimo nome del file in quella directory, a meno che non esista già un file con lo stesso nome.

Esempio:

```
bash$ lfc-ln -s /grid/pippo /grid/pluto
```

La sintassi di `lcg-aa` è livemente più complessa:

```
lcg-aa --vo VO guid lfn
```

‘--vo VO’ Specifica la Virtual Organization da usare

‘guid’ Specifica il GUID del file

‘lfn’ Specifica il LFN da usare. Richiede ‘lfn:’

Esempio:

```
bash$ lcg-aa --vo gridats \
      guid:b0f64a56-0f1e-4ed2-993e-631de0bd06bb \
      lfn:testjpb_rf12
```

5.3 Comandi di uso meno frequente

In questa sezione sono espone alcune informazioni su comandi che richiedono una maggiore dimestichezza con il filesystem di griglia.

Oltre alle operazioni basilari, infatti, è possibile visualizzare e modificare informazioni aggiuntive (metadati) relative ai singoli file, modificare in maniera più raffinata i permessi di accesso e creare repliche di un medesimo file come accennato in [Sezione 5.2 \[Comandi di base\]](#), pagina 50.

5.3.1 Reperire informazioni relative ai file

Questa sezione riporta un elenco di vari comandi che permettono di reperire informazioni utili relative ai file ed agli *Storage Element*.

5.3.1.1 Reperire informazioni sulle Access List

Ad ogni file sono associati sia dei permessi in stile **UNIX** sia delle *Access Control List (ACL)* che possono essere visualizzate o modificate. Le Access List definiscono chi può accedere ad un determinato file o directory e quali permessi ha (scrittura/lettura). Nel caso di directory, inoltre, le ACL permettono di specificare quali saranno le Access List da associare ai nuovi file creati all'interno della stessa.

Il comando `lfc-getacl` permette di visualizzare le ACL relative ad un file o una directory:

```
lfc-getacl path
```

Il risultato del comando precedente su un file apparirà simile al seguente:

```
bash$ lfc-getacl /grid/gridats/crossi/testfile
# file: /grid/gridats/crossi/testfile
# owner: 60125
# group: 60125
user::rwx
group::rwx          #effective:rwx
other::r-x
```

Le prime tre righe mostrano rispettivamente il nome del file, l'utente proprietario ed il gruppo proprietario. Ogni file avrà delle access list impostate di default, che saranno analoghe alle precedenti a meno che non siano state modificate le ACL della directory che lo contiene.

Ogni riga specifica una regola, e può essere relativa ad un utente, un gruppo, o tutti gli altri. Le righe che iniziano per 'user:' o 'group:' possono essere seguite dal nome di un utente o di un gruppo; in tal caso la regola verrà applicata solo all'utente o al gruppo specificato.

In una directory il comando `lfc-getacl` mostrerà anche altre righe:

```
bash$ lfc-getacl /grid/gridats/crossi
# file: /grid/gridats/crossi
# owner: 60125
# group: 60125
user::rwx
group::rwx          #effective:rwx
```

```

other::r-x
default:user::rwx
default:group::rwx
default:other::r-x

```

ovvero alle righe che specificano i permessi dell'utente e del gruppo proprietari, e quelli per tutti gli altri ne esistono altre tre analoghe che specificano con quali permessi verranno creati nuovi file e sottodirectory. Le regole che cominciano con 'default:' hanno una sintassi identica alle altre, ma invece di specificare dei permessi relativi alla directory stessa rappresentano le access list che verranno impostate sui file e le directory che verranno creati all'interno della directory corrente.

5.3.1.2 Recupera il GUID di un file

Il comando `lcg-lg` permette di risalire al GUID di un file conoscendone solo il nome logico od una URL. La sintassi è la seguente:

```
lcg-lg --vo VO lfn_o_surl
```

dove

'--vo VO' specifica la Virtual Organization

'lfn_o_surl'

Specifica il file. Deve essere un protocollo di tipo 'lfn:' o un URL.

Esempio:

```

bash$ lcg-lg --vo
      egrid lfn:/grid/gridats/test.txt
guid:298c666e-9792-4a69-9c5a-6594d755b3eb

```

oppure, conoscendo il URL del file:

```

bash$ lcg-lg -vo gridats
      sfn://egrid-ce-01.egrid.it/storage/gridats/generated/2005-08-22/filec34d1b57-84be-
4898-94a6-eda296a47ebb
guid:298c666e-9792-4a69-9c5a-6594d755b3eb

```

5.3.2 Creare e gestire le repliche

Ricordiamo, cfr. [Sezione 5.1 \[L'indice dei file in griglia\], pagina 49](#), che ogni file ha un unico GUID, ma può essere associato a più URL e più LFN. Ovvero ogni file può risiedere su più SE (ha più URL) e vi si può accedere tramite più nomi logici (ha degli alias). Tali copie fisiche si chiamano *repliche* e possono essere create con il comando `lcg-rep`. Le repliche possono rivelarsi utili nel caso si debba accedere a file piuttosto grossi, e si desidera averne una copia vicino a dove avviene la loro elaborazione.

La sintassi del comando `lcg-rep` è la seguente:

```
lcg-rep --vo VO -d destination src_file
```

dove:

'-d destination'

Si può specificare un URL o semplicemente l'hostname di un SE. Per avere l'elenco degli SE disponibili usare il comando `lcg-infosites`

`--vo VO` Specifica la virtual organization da usare

`src_file`

Specifica il file sorgente. Deve essere specificato anche il protocollo, che può essere `lfn:`, `guid:`, o un URL del tipo `srm:` o `sfn:`.

Supponiamo di avere il file `lfn:/grid/gridats/test.txt`:

```
bash$ lcg-lr --vo \
        gridats lfn:/grid/gridats/test.txt
sfn://egrid-ce-01.egrid.it/storage/gridats/generated/2005
-08-22/filec34d1b57-84be-4898-94a6-eda296a47ebb
```

che risiede sul SE `egrid-ce-01.egrid.it`. Se ne vuole creare una replica sull'host `baciucio.hpc.sissa.it`. Sarà sufficiente dare il comando

```
bash$ lcg-rep --vo gridats \
        -d baciucio.hpc.sissa.it \
        lfn:/grid/gridats/test.txt
```

A questo punto il comando `lcg-lr` ci mostrerà due URL, relative ai due storage element:

```
bash$ lcg-lr --vo gridats lfn:/grid/gridats/test.txt
sfn://baciucio.hpc.sissa.it/storage/gridats/generated/2005-08-22/fi
lea5731c2d-d799-4786-b937-dcce6bae2e02
sfn://egrid-ce-01.egrid.it/storage/gridats/generated/2005-08-22/fi
lec34d1b57-84be-4898-94a6-eda296a47ebb
```

5.3.2.1 Visualizza le repliche di un file

Il comando restituisce una serie di URL relative al file specificato. Si può specificare un LFN, un GUID o un URL. La sintassi è la seguente:

```
lcg-lr --vo VO file
```

dove

`--vo VO` specifica la Virtual Organization

`file` Specifica il file. Deve essere un protocollo di tipo `lfn:`, `guid:` o un URL.

Ad esempio se volessimo conoscere le locazioni fisiche del file `/grid/gridats/test.txt` basterebbe:

```
bash$ lcg-lr --vo gridats \
        lfn:/grid/gridats/test.txt
sfn://egrid-ce-01.egrid.it/storage/gridats/generated/2005
-08-22/filec34d1b57-84be-4898-94a6-eda296a47ebb
```

Se invece si conoscesse già il GUID del file allora basterebbe:

```
bash$ lcg-lr --vo gridats \
        guid:298c666e-9792-4a69-9c5a-6594d755b3eb
sfn://egrid-ce-01.egrid.it/storage/gridats/generated/2005
-08-22/filec34d1b57-84be-4898-94a6-eda296a47ebb
```

Nel caso il file abbia più repliche (cfr [Sezione 5.3.2 \[Creare e gestire le repliche\]](#), [pagina 57](#).) il risultato mostrerà tutte le URL relative alle varie repliche. Nell'esempio seguente viene creata una replica e dato nuovamente il comando `lcg-lr`.

```

bash$ lcg-rep --vo gridats \
        -d baciuco.hpc.sissa.it \
        lfn:/grid/gridats/test.txt \
bash$ lcg-lr --vo gridats lfn:/grid/gridats/test.txt
sfn://baciuco.hpc.sissa.it/storage/gridats/generated/2005-08-
22/filea5731c2d-d799-4786-b937-dcce6bae2e02
sfn://egrid-ce-01.egrid.it/storage/gridats/generated/2005-08-
22/filec34d1b57-84be-4898-94a6-eda296a47ebb

```

5.3.2.2 Visualizza gli alias di un file

Gli alias, come detto in [Sezione 5.2.9 \[Creare più nomi logici per il medesimo file\]](#), pagina 55, sono semplicemente più nomi logici che corrispondono allo stesso file, e sono visibili con il comando `lfc-ls -l` direttamente dal catalogo, ma tale comando è, appunto, limitato alla struttura del catalogo, perciò è possibile visualizzare esclusivamente *se un file è un alias* e nel caso *a quale file punta un alias*.

Il comando `lcg-la` mostra invece un elenco degli alias relativi al file specificato. La sintassi è la seguente:

```
lcg-la --vo VO file
```

dove

‘--vo VO’ specifica la Virtual Organization

‘file’ Specifica il file. Deve essere un protocollo di tipo ‘lfn:’, ‘guid:’ o un URL.

Esempio:

```

bash$ lfc-ls -l /grid/gridats/crossi/
lrwxrwxrwx 1 60125 60125 0 Aug 22 17:02 test-alias.txt -> /grid/gridats/
crossi/test.txt
-rwxrwxr-x 1 60125 60125 0 Aug 22 17:01 test.txt
bash$ lcg-la --vo gridats \
        lfn:/grid/gridats/crossi/test.txt
lfn:/grid/gridats/crossi/test.txt
lfn:/grid/gridats/crossi/test-alias.txt

```

Appendice A Codice degli esercizi di sottomissione

Qui di seguito sono riportati i programmi usati nel secondo e terzo esempio di sottomissione (Vedi [Sezione 4.3.1 \[Il programma\]](#), pagina 30, Vedi [Sezione 4.4.1 \[Un esempio MPI\]](#), pagina 41.); questi vengono forniti al solo scopo di essere usati nelle esercitazioni e sono stati concepiti per essere più semplici possibile e non come esempi di efficienza o di buona programmazione.

Anche se il codice è stato abbondantemente commentato, la sua comprensione non è necessaria per seguire il tutorial.

A.1 area.c

Il programma va compilato con `gcc -Wall -o area area.c -static`.

```
#include <stdlib.h>
#include <stdio.h>

/*
  Semplice programma che calcola l'area di una circonferenza.

  La necessita' di calcolare l'errore nel risultato viene mitigata
  dal fatto che lo si conosce gia'.

  Il programma sfrutta la simmetria della circonferenza per eseguire
  la simulazione solo nel I quadrante (x e y >= 0).
*/

int main(int argc, char *argv[])
{
    FILE *f;
    double raggio;      /* il raggio della circonferenza */
    unsigned int seme;  /* il seme del generatore random */
    int N;              /* il numero di coppie da generare in totale */

    /* Apri il file con il raggio */
    f=fopen("raggio.txt", "r");
    if(f==NULL)
        exit(1);
    /* Leggine il contenuto */
    fscanf(f, "%lf", &raggio);
    /* Stampalo a schermo */
    printf("Raggio della circonferenza: %g\n", raggio);
    /* Chiudi il file */
    fclose(f);

    /* Apri il file col seme random */
    f=fopen("seme.txt", "r");
    if(f==NULL)
        exit(1);
    /* Leggine il contenuto */
    fscanf(f, "%u", &seme);
    /* Stampalo a schermo */
    printf("Seme random: %u\n", seme);
    /* Chiudi il file */
    fclose(f);
}
```

```

/* La lunghezza della simulazione */
N=atoi(argv[1]);

/* Inizializza il generatore di numeri pseudocasuali */
srand(seme);

/* Inizia la simulazione*/
{
    double x,y;          /* Coordinate del punto */
    double area;          /* l'area stimata */
    int hit=0;            /* coppie interne alla circonferenza */
    int i;                /* contatore */

    /* Apri il file per l'output */
    f=fopen("output.txt","w");

    for(i=0;i<N;i++) {

        /* Crea una coppia di numeri in [0,raggio[x[0,raggio[*/
        x=(raggio*rand())/(1.0+RAND_MAX);
        y=(raggio*rand())/(1.0+RAND_MAX);

        /* Se (x,y) e' nel quarto di circonferenza, aggiorna il contatore */
        if(x*x+y*y<=raggio*raggio)
            hit=hit+1;

        /* Stampa il risultato nel file ogni 10000 cicli*/
        if(i%10000==9999) {
            /* Calcola l'area stimata */
            area=4*(raggio*raggio*hit)/i;
            fprintf(f,"%d %d %20.15g\n",hit,i,area);
        }
    }

    /* Chiudi il file di output */
    fclose(f);
}

/* Esci dal programma*/
return 0;
}

```

A.2 areaMPI.c

Il programma va compilato con le librerie MPICH e architettura **p4.ch** per poter essere lanciata in griglia.

Se le librerie sono correttamente installate, è sufficiente lanciare:

```
mpicc -o areaMPI areaMPI.c
```

Per testare il programma su una macchina sola, digitare¹:

¹ dopo aver creato un file di configurazione chiamato 'params.txt' contenente il seme random, il raggio e il numero di punti, tipo:

```

15
2.3
100000000

```

```

mpirun -np 1 ./areaMPI
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>

/*
    Semplice programma che calcola l'area di una circonferenza.

    La necessita' di calcolare l'errore nel risultato viene mitigata
    dal fatto che lo si conosce gia'.

    Il programma sfrutta la simmetria della circonferenza per eseguire
    la simulazione solo nel I quadrante (x e y >= 0).
*/

int main(int argc, char *argv[])
{
    FILE *f=NULL;
    double raggio;          /* il raggio della circonferenza */
    unsigned int seme;      /* il seme del generatore random */
    unsigned long long int N; /* il numero di coppie da generare in totale */

    int me;                 /* L'indice del processore corrente */
    int cpu;                /* Numero di processori disponibili */
    int r;                  /* Valore tornato dalle f. MPI */

    unsigned long long int *collected; /* Array temporaneo per raccogliere
                                         i risultati */

    /*  /* solo per debugging... */  */
    /*  setbuf(stdout, NULL);  */

    /* Inizializza il sistema MPI */
    if((r=MPI_Init(&argc,&argv))!=MPI_SUCCESS)
        MPI_Abort(MPI_COMM_WORLD,r);

    /* Ottieni il numero di processori */
    if((r=MPI_Comm_size(MPI_COMM_WORLD,&cpu))!=MPI_SUCCESS)
        MPI_Abort(MPI_COMM_WORLD,r);

    collected=(long long int *) malloc(sizeof(long long int)*cpu);
    if(collected==NULL) {
        MPI_Finalize();
        return -1;
    }

    /* Ottieni il numero del processore corrente */
    if((r=MPI_Comm_rank(MPI_COMM_WORLD,&me)))
        MPI_Abort(MPI_COMM_WORLD,r);

    /* Esegui queste operazioni solo sul processore 0 */

```

```

if(me==0) {
    /* Apri il file con i dati */
    f=fopen("params.txt","r");
    if(f==NULL)
        exit(1);
    /* Leggine il contenuto */
    fscanf(f,"%u",&seme);
    fscanf(f,"%lf",&raggio);
    fscanf(f,"%llu",&N);

    /* Stampa i dati a schermo */
    printf("Seme random: %u\n",seme);
    printf("Raggio della circonferenza: %g\n",raggio);
    printf("Iterazioni: %llu\n",N/cpu);

    /* Chiudi il file */
    fclose(f);
}

/* Passa il raggio */
if((r=MPI_Bcast (&raggio,1,MPI_DOUBLE,0,MPI_COMM_WORLD))!=MPI_SUCCESS)
    MPI_Abort(MPI_COMM_WORLD,r);

/* Passa il seme random */
if((r=MPI_Bcast (&seme,1,MPI_INT,0,MPI_COMM_WORLD))!=MPI_SUCCESS)
    MPI_Abort(MPI_COMM_WORLD,r);

/* Passa la lunghezza */
if((r=MPI_Bcast (&N,1,MPI_LONG_INT,0,MPI_COMM_WORLD))!=MPI_SUCCESS)
    MPI_Abort(MPI_COMM_WORLD,r);

N/=cpu;

/* Inizializza il generatore di numeri pseudocasuali */
srand(seme+me);

/* Inizia la simulazione*/
{
    long double x,y;          /* Coordinate del punto */
    long double area;         /* l'area stimata */

    unsigned long long int hit=0;    /* coppie interne alla circonferenza */
    unsigned long long int i;        /* contatore */

    /* Apri il file per l'output */
    if(me==0)
        f=fopen("output.txt","w");

    for(i=0;i<N;i++) {

        /* Crea una coppia di numeri in [0,raggio[x[0,raggio[*/
        x=(raggio*rand()/(1.0+RAND_MAX));
        y=(raggio*rand()/(1.0+RAND_MAX));

        /* Se (x,y) e' nel quarto di circonferenza, aggiorna il contatore */
    }
}

```

```

        if(x*x+y*y<=raggio*raggio)
            hit=hit+1;

        /* Stampa il risultato nel file ogni 10000 cicli*/
        if(i%10000==9999) {
            unsigned long long int tothit=0;
            int j;

            /*
             Raccogli i dati dagli altri processori
             Reduce non funziona su long long int: la
             somma va fatta a mano.
            */
            if((r=MPI_Allgather(&hit,2,MPI_INT,
                               collected,2,MPI_INT,
                               MPI_COMM_WORLD))!=MPI_SUCCESS)
                MPI_Abort(MPI_COMM_WORLD,r);

            for(j=0;j<cpu;j++)
                tothit+=collected[j];

            /* Calcola l'area stimata */
            area=4*(raggio*raggio*tothit)/(i*cpu);
            if(me==0)
                fprintf(f,"%llu %llu %20.20Lg\n",tothit,i*cpu,area);
        }
    }

    /* Chiudi il file di output */
    if(me==0)
        fclose(f);
}

free(collected);
MPI_Finalize();

/* Esci dal programma*/
return 0;
}

```

Appendice B Referenze

TODO con riferimenti a tutti i documenti utili, con link

Indice

-

'--dir', opzione di <code>edg-job-get-output</code>	26
'--help', opzione per i comandi <code>edg-job-*</code>	26
'--input' ('-i') con i comandi <code>edg-job-*</code>	31
'--noint' con i comandi <code>edg-job-*</code>	31
'--output' ('-o') con i comandi <code>edg-job-*</code>	31
'--static'	23

.

'globus', directory con i certificati	13
---	----

A

<i>Access Control List</i>	56
ACL	56
Alias, creazione	55
'area.c', descrizione	30
'area_base.jdl'	30
'areaMPI.c', descrizione	41
'areaMPI.jdl'	42
'areaMPI.sh'	42
'Arguments'	24
autenticazione, concetti di base	12
autenticazione, i certificati	12

B

BDII	3, 22
<i>Berkeley Database Information Index</i>	3
<i>Berkley Database Information Service</i>	22

C

CA (<i>Certification Authority</i>)	12
cancellazione di un job	26
caricamento dati in un job	45
catalogo	3
<i>Catalogo</i>	49
CE	3, 22
certificati	12
certificato	3
certificato proxy	12
'check_status.sh'	37
chiave privata	12
chiave pubblica	12
chiavi private e pubbliche	13
coda, descrizione	22
code, differenze tra tipi di	42
<code>computeF</code>	29
<i>Computing Element</i>	3, 22
configurazione data e ora	6
configurazione egrid	7
'Current Status', output di <code>edg-job-status</code> ..	25

D

'DataAccessProtocol'	45
dati, caricamento dati in un job dalla griglia ...	45
dati, salvataggio dati da un job alla griglia	45

E

<code>edg-job-cancel</code> , command	26, 31
<code>edg-job-get-logging-info</code> , command	27
<code>edg-job-get-output</code> , command	26, 31
<code>edg-job-status</code> , command	25, 31
<code>edg-job-submit</code> , command	24, 31
<code>egrid-infosites</code>	50
'Executable'	24

F

'file_operations.jdl'	46
'file_operations.sh'	46
FQAN	14

G

gcc: compilazione per la griglia	30
'get_results.sh'	38
<i>Grid Resource Information Service</i>	3
<i>Grid Unique Identifier</i>	49
Grid, componenti principali	2
Grid, concetti di base	2
Grid, implementazione	2
<code>grid-change-pass-phrase</code> , comando	17
griglia, tipi di installazione	4
GRIS	3
GUID	49

H

hardware, requisiti minimi	4
'Hello.jdl'	24
hostname, configurazione	6

I

imbarazzantemente parallelo, esempio di job ...	29
'info.jdl'	27
'info.sh'	27
'InputData'	45
'InputSandbox'	24

J

JA	22
JC	22
JDL, caricamento dei dati dalla griglia	45
JDL, eseguire comandi di shell in griglia	27
JDL, formato e sintassi	23
JDL, 'Hello World!'	24
JDL, per jobs MPI	42
JDL, salvataggio dei risultati in griglia	45
JDL, tramite wrapping	34
<i>Job Description Language</i>	23
<i>Job ID</i>	24
Job ID, salvataggio e caricamento in un file	31
job seriale	29
Job Status	25
job, caricamento dei dati dalla griglia	45
job, esempio di accesso ai dati in griglia	46
job, salvataggio dei risultati in griglia	45
Jobs, automatizzare il controllo sullo stato	37
Jobs, automatizzare il ritiro dei dati	38
Jobs, automatizzare la sottomissione	31
jobs, compilazione degli eseguibili	23
jobs, tipi supportati	23
Jobs, Uso di template	32
'JobType'	42
<i>Jop Adapter</i>	22
<i>Jop Controller</i>	22

K

<i>keyword</i> (in un file JDL	23
--------------------------------------	----

L

LBS	22
LCG	2
lcg-aa	55
lcg-cp	53
lcg-cp, eseguito da un job	45
lcg-cr	52
lcg-del	53
lcg-infosites	52
lcg-infosites, esempio pratico per la sottomissione	46
lcg-lg, comando	57
lcg-lr, comando	58
<i>lcpbs</i>	42
'lcpbs_MPI.sh'	42
lfc-ln	55
lfc-ls	50
lfc-mkdir	54
lfc-rename	54
lfc-rm	55
LFN	49
Live CD, avvio	5
Live CD, configurazione	5
Live CD, configurazione rete	5
Live CD, descrizione	4

Live CD, download	4
Live CD, installazione	8
Live CD, masterizzazione	5
LM	22
<i>Log Monitor</i>	22
<i>Logging and Booking Service</i>	22
<i>Logical File Name</i>	49
'LogicalFileName'	45

M

<i>matchmaking</i>	22
middleware	2
MPI, librerie necessarie	42
MPICH	42
MyProxy, introduzione	17
myproxy-get-delegation, comando	20
myproxy-info	18
myproxy-info, comando	20
myproxy-init, generazione certificato myproxy rinnovabile, comando	19
myproxy-init, generazione certificato myproxy, comando	18
MYPROXY_SERVER	17

N

'NodeNumber'	42
non interattivi, comandi <i>edg-job-</i>	31

O

'output.txt'	30
'OutputData'	45
'OutputSandbox'	24

P

parametri variabili in un JDL	32
<i>pbs</i>	42
prerequisiti	1
proxy	3
proxy, cambiare passphrase	17
proxy, certificato	13
proxy, creazione	15
proxy, distruzione	16
proxy, informazioni sul certificato attuale	16

Q

queue	22
-------------	----

R

RA (<i>Registration Authority</i>)	13
'raggio.txt'	30
RB	22
redirezione dell'input e output di un job	24
<i>Replica Location Service</i>	3
'Requirements'	42
<i>Resource Broker</i>	22
rete, configurazione	5, 6
ritiro dei risultati di un job	26
Ritiro di un job, automatizzazione	38
RLS	3

S

<i>Sandbox</i>	24
scheduling	22
SE	3, 49
SE, avvicinamento ad un	45
'seme.txt'	30
seriali, job	23
<i>Site URL</i>	49
sottomissione	22
Sottomissione	24
sottomissione di un job in griglia	24
sottomissione, automatizzazione	31
sottomissione, introduzione	22
stato di un job	25
status di un job, automatizzazione	37
'Status Reason', output di <code>edg-job-status</code>	25
'StdError'	24
'StdOutput'	24
<i>Storage Element</i>	3, 49
'StorageElement'	45
SURL	49

T

template	32
'template.jdl'	32
'template_submission.jdl'	32

U

UI	2
<i>UI in userspac</i>	9
UI, configurazione	7
UI, configurazione hostname	6
UI, configurazione rete	5
UIUS	9
UIUS, cambiare la configurazione	11
UIUS, configurazione	9
UIUS, disinstallazione	11
UIUS, download	9
<i>User Interface</i>	2
'usercert.pem'	12
'userkey.pem'	12

V

<i>Virtual Organization</i>	3
VO	3
VOMS, attributi	14
VOMS, estensioni	14
<code>voms-proxy-destroy</code> , comando	16
<code>voms-proxy-info</code> , comando	16
<code>voms-proxy-init</code> , comando	15

W

WN	3, 22
<i>Worker Node</i>	3, 22
wrapper	34
'wrapper.jdl'	34
'wrapper.sh'	34
'wrapper_submit.sh'	34

Indice dei comandi

E

edg-job-cancel.....	26, 31
edg-job-get-logging-info	27
edg-job-get-output	26, 31
edg-job-status.....	25, 31
edg-job-submit.....	24, 31
egrid-infosites	50

G

grid-change-pass-phrase	17
-------------------------------	----

L

lcg-aa.....	55
lcg-cp.....	53
lcg-cr.....	52
lcg-del.....	53
lcg-infosites	52
lcg-lg.....	57

lcg-lr	58
lfc-ln.....	55
lfc-ls.....	50
lfc-mkdir	54
lfc-rename	54
lfc-rm.....	55

M

myproxy-get-delegation	20
myproxy-info	20
myproxy-init, generazione certificato myproxy	18
myproxy-init, generazione certificato myproxy rinnovabile.....	19

V

voms-proxy-destroy.....	16
voms-proxy-info	16
voms-proxy-init	15

Sommario abbreviato

1	Scopo di questo manuale	1
2	Concetti di base.	2
3	Preparazione dell'ambiente di lavoro (User Interface)	4
4	Sottomettere un programma in griglia	22
5	Gestire i file in griglia	49
A	Codice degli esercizi di sottomissione	60
B	Referenze	65
	Indice	66
	Indice dei comandi	69