ISTITUTO NAZIONALE DI FISICA NUCLEARE
Sezione di Milano Bicocca, Gruppo Collegato di Parma

INFN/CCR-08/04
22, Dicembre 2008

CCR-30/2008/P

INTEGRARE IL CALCOLO LOCALE MPI NELLA FARM INFN-GRID

Roberto Alfieri, Roberto Covati e Enrico Tagliavini

INFN-Sezione di Milano Bicocca, Gruppo Collegato di Parma
e Università degli Studi di Parma
Viale G.P. Usberti n.7/A I-43100 Parma, Italy

Abstract

Si descrive l'esperienza di integrazione dei *cluster* locali utilizzati per il calcolo MPI presso il Gruppo Collegato di Parma nella *farm* Grid. Tale integrazione è stata realizzata in modo tale da garantire un accesso prioritario e trasparente alle risorse per gli utenti locali ma nel caso in cui tali risorse risultino inutilizzate è stato previsto il loro riutilizzo per *job* MPI provenienti dagli utenti di InfnGrid.
Vengono inoltre analizzate le problematiche legate all'introduzione del *tool mpi-start* per la gestione contemporanea di diversi *flavor* di MPI.

Pubblicato da ***SIS–Pubblicazioni***
Laboratori Nazionali di Frascati

INTRODUZIONE

I centri di calcolo che negli ultimi anni hanno iniziato a supportare il *Grid Computing* avevano già, in molti casi, un complesso insieme di installazioni tradizionali eterogenee, con *farm* dotate di diversi ambienti *software* e diverse politiche amministrative.

L'eventuale supporto di MPI introduce nuove variabili nelle configurazioni delle *farm*, quali ad esempio i diversi *flavor* esistenti di MPI, le diverse possibili infrastrutture di comunicazione ad alta velocità (*Gigabit Ethernet*, *Infiniband*, *Myrinet*, etc.), la condivisione dei dati dell'utente tra i nodi, etc. Il *tool mpi-start*, recentemente integrato anche nel *middleware* di Egee, rende trasparente l'utilizzo di MPI per l'utente, ma complica ulteriormente il lavoro dell'amministratore del sito.

In questo documento vogliamo riportare l'esperienza acquisita presso il sito INFN-Parma in merito al progetto di razionalizzazione della gestione e dell'utilizzo delle risorse di calcolo mediante l'integrazione dei *cluster* locali nell'infrastruttura di InfnGrid. È stato perseguito l'obiettivo di rendere disponibili le risorse agli utenti Grid in caso di inutilizzo locale.

Il documento si soffermerà in particolare sugli aspetti di configurazione e utilizzo di MPI per cui si vuole mantenere il supporto *multi-flavor* per l'accesso sia locale sia Grid.

I CLUSTER DA INTEGRARE

Presso l'INFN di Parma sono attivi diversi gruppi di ricerca che si occupano di simulazione numerica, particolarmente nei campi della Relatività Generale e della Teoria di Gauge su Reticolo.

Per quanto riguarda la simulazione numerica della Relatività Generale, la Sezione fa parte di una collaborazione nazionale che ha installato a Parma due *cluster* MPI:

- **Albert100**, installato nel 2002, è composto da 44 *Dual* PIII a 1.13 GHz, con interconnessioni in *Fast Ethernet*;
- **Albert2**, installato nel 2004, è dotato di 16 *Dual Opteron* e *Infiniband*.

Le simulazioni su reticolo della QCD vengono svolte prevalentemente con gli strumenti di calcolo dei progetti APE, con cui Parma collabora attivamente, ma anche con *cluster* di PC. Attualmente vi sono due installazioni:

- **HAM**, composto da 10 *Dual Athlon* a 2GHz con rete *Myrinet*;
- **Exalibur**, dotato di un *blade* con 7 *Dual Xeon.*

Esiste inoltre una *farm* di Calcolo Dipartimentale, senza supporto MPI, con 4 nodi di calcolo a 64 bit e 3 nodi a 32 bit, a disposizione di tutti gli utenti locali del sito.

Infine, dal 2004, è attivo un sito **INFN-Grid** con 4 *Dual Xeon* 2.4GHz pensato prevalentemente per il calcolo MPI e per lo *storage* di dati delle *Virtual Organization* di Theophys e ILDG (*International Lattice Data Grid*).

Il *Resource Manager* utilizzato da tutti i *cluster* (che ne fanno uso) è OpenPBS o Torque/Maui.

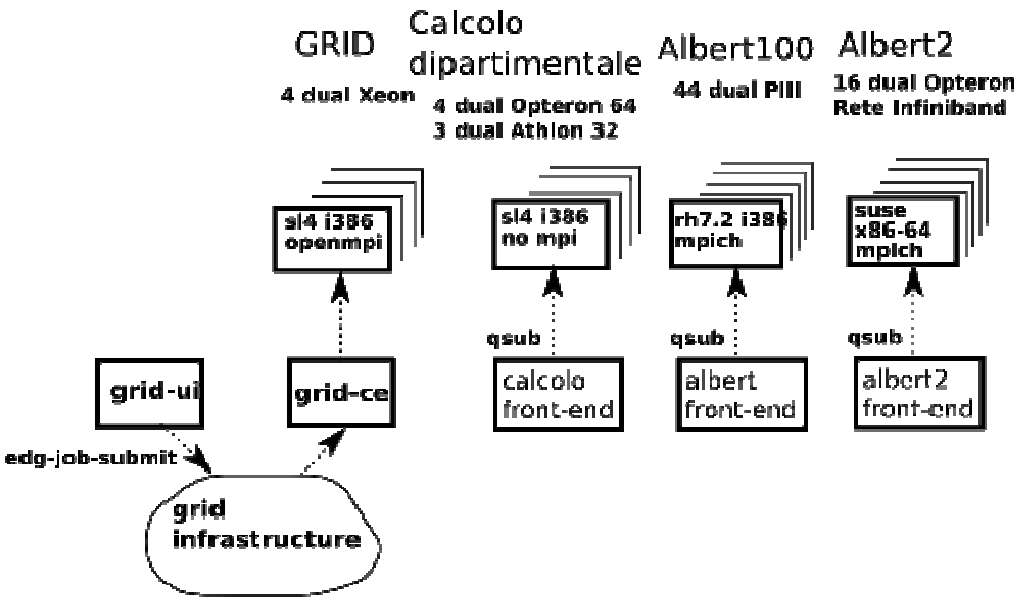

FIG. 1: I *cluster* da integrare.

IL PROGETTO

L'obiettivo del progetto è stato creare una infrastruttura comune in grado di integrare progressivamente tutti i *cluster* MPI nella *farm* Infn-Grid. Inizialmente sono stati integrati i 3 *cluster* Albert100, Albert2 e la *farm* per il Calcolo Dipartimentale, reinstallandoli con *Scientific Linux* 4 (SL4) che è la distribuzione attualmente supportata da gLite. I *cluster* a 32 bit vengono installati con la distribuzione SL4_i386, mentre per i *cluster* a 64 bit viene utilizzata la SL4_x86-64. La piattaforma a 64, non ancora supportata da gLite, potrà essere utilizzata inizialmente solo dagli utenti locali. Un importante obiettivo consiste nel mantenere per tutti i *cluster* l'accesso locale prioritario, possibilmente con gli stessi strumenti e *flavor* MPI utilizzati in precedenza.

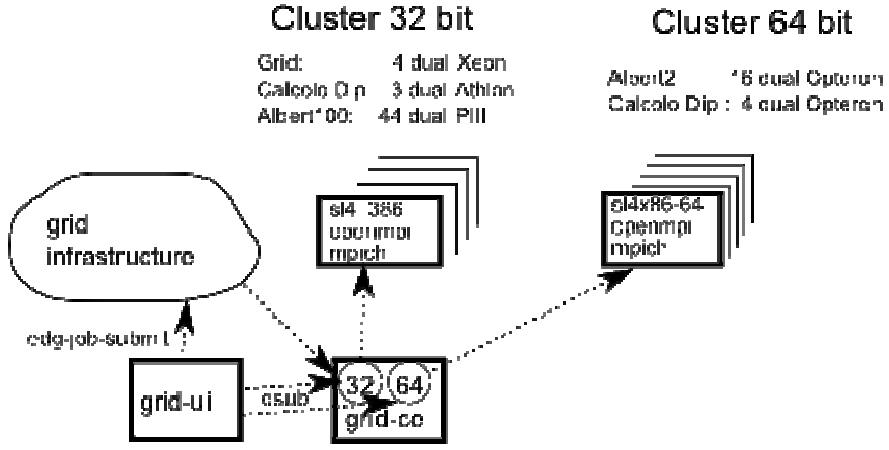

FIG. 2: Il progetto di integrazione.

In questo progetto si è voluta concentrare l'attenzione sugli aspetti di integrazione legati ad MPI e per questo sono state tralasciate le problematiche di *Access Policy* relative alla configurazione di *scheduling* delle code per la gestione delle priorità e del *fair sharing*.

IL SUPPORTO DI MPI IN EGEE

Storicamente il *middleware* di Egee supporta esplicitamente l'implementazione MPICH di MPI mediante le direttive *JobType* che deve assumere il valore MPICH. Un tipico *job ClassAd* ha il seguente aspetto:

```
JobType = ”MPICH”;
Executable = ”mio-job”;
NodeNumber = 8;
...
```

In questo esempio “mio-job” è inteso come l'eseguibile MPI, compilato con MPICH, che il *middleware* metterà in esecuzione (mpirun mio-job -np 8 [...]) sui nodi assegnati dallo *scheduler*. Il sito che supporta MPI deve semplicemente pubblicare il *Tag* MPICH.

Il modello e' semplice ma eccessivamente rigido poiché non prevede l'utilizzo di diversi *flavor* di MPI (OpenMPI, LAM, etc.) e non consente l'utilizzo di *script* di pre/post esecuzione, utili ad esempio per la compilazione remota o la replica dei file quando la *home* non e' condivisa tra i nodi di calcolo.
Questa architettura e' stata superata con l'*update* 14 di gLite3.1[1] (Febbraio 2008) che introduce alcune modifiche sostanziali nel supporto di MPI, recependo le indicazioni dell'**MPI *Working Group* di Egee**[2]. Le principali novità consistono nell'utilizzo del *tool mpi-start* e nell'installazione sui nodi di calcolo di uno *script mpirun* “dummy”.

Mpi-start

Sviluppato all'**HLRS di Stoccarda**[3], *mpi-start* e' un insieme di *script* che consente di gestire in modo flessibile e automatico la configurazione e l'utilizzo di un sito MPI. Il *tool* è organizzato in 3 *framework*:

- ***Scheduler Framework*:** consente di estrarre automaticamente il *Machinefile* dallo *scheduler*. Gli *scheduler* supportati sono SGE, PBS e LSF.
- **MPI *Framework*:** gestisce la catena generica di esecuzione *pre_run_hook/mpi_exec/post_run_hook* in cui l'utente può indicare il *flavor* di MPI da utilizzare. Sono supportati OpenMPI, MPICH, MPICH2 e Lam-MPI.
- ***File Distribution Framework***: questo *framework* si occupa della distribuzione dei *file* sui WN quando la *directory* di lavoro non e' condivisa. I protocolli utilizzati sono SSH (*password-less*) e *mpi_mt*.

Per utilizzare questo *tool* l'utente mette in esecuzione un *wrapper*, che qui chiameremo ***mpi-start-wrapper.sh***, il quale definisce alcune variabili d'ambiente (nome dell'eseguibile

---

1 http://glite.web.cern.ch/glite/packages/R3.1/updates.asp
2 http://www.grid.ie/mpi/wiki/FrontPage
3 http://www.hlrs.de/organization/amt/projects/mpi-start/

MPI, *flavor* MPI da utilizzare, etc.) e quindi avvia *mpi-start*. Ecco un esempio semplificato di *script*:

```
export I2G_MPI_APP=test-mpi
export I2G_MPI_TYPE=openmpi
export I2G_MPI_PRE_RUN_HOOK=mpi-hooks.sh
export I2G_MPI_POST_RUN_HOOK=mpi-hooks.sh
# Invoke mpi-start:
$I2G_MPI_START
```

Le fasi di *pre-execution* e *post-processing* possono essere personalizzate dall'utente mediante le due funzioni *PRE_RUN_HOOK* e *POST_RUN_HOOK*. Nel seguente file di esempio *mpi-hooks.sh* il sorgente viene compilato in *pre-execution*:

```
#!/bin/sh
pre_run_hook ()  {mpicc -o ${I2G_MPI_APP} ${I2G_MPI_APP}.c  }
post_run_hook () {}
```

***mpirun* "dummy"**

Per poter eseguire lo *script mpi-start-wrapper.sh*, al posto del compilato MPICH, e' necessario un *workaround* che consiste nella installazione di un opportuno *script* ***mpirun*** (*dummy*). Questo *script* verrà installato da Yaim sui WN in /opt/glite/bin/ e dovrà precedere altri eventuali eseguibili "mpirun" nella variabile d'ambiente *PATH* del WN.

INSTALLAZIONE E CONFIGURAZIONE

MPI è ora supportato da Yaim grazie al modulo ***glite-yaim-mpi***[4] che deve essere installato e configurato sia sul CE sia sui WN. Il modulo è installato sui WN dal *metapackage glite-MPI_utils*, assieme a MPICH (attualmente la versione 1.2.7) e ad *i2g-mpi-start* (attualmente la versione 0.0.52). **MPICH** è il *flavor* MPI ufficialmente supportato da gLite. Viene distribuito con binari installati in /opt/mpich-1.2.7p1/bin/. L'installazione di ulteriori *flavor* deve essere gestita manualmente.

OpenMPI[5] è una recente implementazione che integra diversi altri progetti (FT-MPI, LA-MPI, LAM, e PACX-MPI) e si sta diffondendo rapidamente tra gli utenti MPI.

La distribuzione SL4x include il pacchetto ***openmpi*** (attualmente 1.2.3) che però installa i binari in /usr/bin (e quindi in conflitto con l'*mpirun* "dummy" che verrà installato da Yaim in /opt/glite/bin/).

Volendo utilizzare OpenMPI è quindi necessario modificare il prefisso di installazione. Per la ricompilazione[6] bisogna installare il sorgente ed eseguire un comando del tipo:

---

[4] http://www.grid.ie/mpi/wiki/YaimConfig

[5] http://www.open-mpi.org/

[6] Sul nodo dove avviene la ricompilazione è necessario che il pacchetto ***torque-devel*** sia installato per avere il supporto per l'*autodiscovery* del *Machinefile* e del numero di processori. Inoltre la versione 0.0.52 di *i2g-mpi-start* contiene un *bug* nel supporto ad OpenMPI (nel file /opt/i2g/etc/mpi-start/openmpi.mpi); è quindi necessario aggiornare manualmente *mpi-start* ad una versione più recente (ad esempio la 0.0.58).

```
rpmbuild -bb --define 'mflags -jN' --define 'build_all_in_one_rpm 1' --
define 'install_in_opt 1' --define 'cflags -g' --target <PLATFORM>
/usr/src/redhat/SPECS/openmpi-1.2.6.spec
```

Per la configurazione di Yaim è necessario aggiungere le opportune direttive in ***ig-site-info.def*** come indicato nei file di esempio in **/opt/glite/yaim/examples/siteinfo/services/**.

Ad esempio se si desidera una installazione di OpenMPI 1.2.6 e MPICH 1.2.7 con *shared-home* la configurazione necessaria è la seguente:

```
MPI_MPICH_ENABLE=${MPI_MPICH_ENABLE:-"yes"}
MPI_MPICH_PATH="/opt/mpich-1.2.7p1/"
MPI_MPICH_VERSION="1.2.7"
MPI_MPICH_MPIEXEC="/opt/mpiexec-0.82/bin/mpiexec"
MPI_OPENMPI_ENABLE=${MPI_OPENMPI_ENABLE:-"yes"}
MPI_OPENMPI_PATH="/opt/openmpi/1.2.6/"
MPI_OPENMPI_VERSION="1.2.6"
MPI_OPENMPI_MPIEXEC="/opt/openmpi/1.2.6/bin/mpiexec"
# se la home è condivisa dai WN per esempio con nfs:
MPI_SHARED_HOME="yes"
# se è abilitata l'autenticazione ssh sui WN:
MPI_SSH_HOST_BASED_AUTH="yes"
```

Queste impostazioni vengono definite come variabili d'ambiente sui WN e possono essere verificate con il comando:

```
env|grep MPI_
```

Inoltre vengono automaticamente pubblicate come *Tag* di *CE_RUNTIMEENV*.

Nell'esempio precedente i *Tag* pubblicati sono:

```
MPICH, MPICH-1.2.7p1, OPENMPI, OPENMPI-1.2.6, MPI-START e MPI_SHARED_HOME
```

Se si desidera usare più di una implementazione/versione di MPI si devono definire le corrispondenti variabili di configurazione (ovvero *MPI_<flavor>_ENABLE*, etc.).

Inoltre, se i WN e il CE hanno la *home* condivisa, è consigliato l'utilizzo di PBS come *Job Manager* (anziché lcgpbs).

Il comando per configurare Yaim sul WN è il seguente:

```
ig_yaim -c -s ig-site-info.def -n ig_WN_<nodetype> -n MPI_WN
```

Questo comando genera, tra le altre cose, lo *script dummy* /opt/glite/bin/mpirun
E' quindi fondamentale che nella variabile d'ambiente *PATH* /opt/glite/bin preceda i *path* delle eventuali implementazioni/versioni di MPI.

ESECUZIONE IN GRID

mpi-start-wrapper

Questo *script* ha il compito di definire le variabili d'ambiente necessarie a *mpi-start* per il *flavor* che si intende usare. Definisce inoltre gli *script hook* scelti. In ultimo lo *script* lancia *mpi-start*.

```
#!/bin/bash
# Pull in the arguments.
MY_EXECUTABLE=`pwd`/$1
MPI_FLAVOR=$2
# Convert flavor to lowercase for passing to mpi-start.
MPI_FLAVOR_LOWER=`echo $MPI_FLAVOR | tr '[:upper:]' '[:lower:]'`
# Pull out the correct paths for the requested flavor.
eval MPI_PATH=`printenv MPI_${MPI_FLAVOR}_PATH`
# Ensure the prefix is correctly set.  Don't rely on the defaults.
eval I2G_${MPI_FLAVOR}_PREFIX=$MPI_PATH
export I2G_${MPI_FLAVOR}_PREFIX
 # Touch the executable.
 #It exist must for the shared file system check.
# If it does not, then mpi-start may try to distribute the executable
# when it shouldn't.
touch $MY_EXECUTABLE
# Setup for mpi-start.
export I2G_MPI_APPLICATION=$MY_EXECUTABLE
export I2G_MPI_APPLICATION_ARGS=
export I2G_MPI_TYPE=$MPI_FLAVOR_LOWER
export I2G_MPI_PRE_RUN_HOOK=mpi-hooks.sh
export I2G_MPI_POST_RUN_HOOK=mpi-hooks.sh
# If these are set then you will get more debugging information.
export I2G_MPI_START_VERBOSE=1
#export I2G_MPI_START_DEBUG=1
# Invoke mpi-start.
$I2G_MPI_START
```

Lo *script* è molto generale ed è modificabile per le specifiche esigenze dell'utente. Per esempio se si vuole utilizzare questo *script* su un *cluster* in cui non è presente *mpi-start* basta aggiungere all'inizio del precedente *script*

```
if [ "x$I2G_MPI_START" = "x" ]; then
    # untar mpi-start and set up variables
    tar xzf mpi-start-*.tar.gz
    export I2G_MPI_START=bin/mpi-start
    MPIRUN=`which mpirun`
    export MPI_MPICH_PATH=`dirname $MPIRUN`
fi
```

Hook per *mpi-start*

Si devono definire le due funzioni *pre_run_hook* e *post_run_hook*, anche in *file* diversi, che verranno chiamate da *mpi-start*.

Ecco un esempio di *script hook*. Questo *script* gestisce le operazioni di pre e post *run*, come ad esempio compilare il programma MPI o copiare i risultati su uno *Storage Element*, ed è essenziale che sia un *file* separato dal precedente *script wrapper*:

```
#!/bin/sh

# This function will be called before the MPI executable is started.
# You can, for example, compile the executable itself.
#
```

```
    pre_run_hook () {
      # Compile the program.
      echo "Compiling ${I2G_MPI_APPLICATION}"
      # Actually compile the program.
     cmd="mpicc    ${MPI_MPICC_OPTS}    -o    ${I2G_MPI_APPLICATION}    $
{I2G_MPI_APPLICATION}.c"
      echo $cmd
      $cmd
      if [ ! $? -eq 0 ]; then
        echo "Error compiling program.  Exiting..."
        exit 1
      fi
      # Everything's OK.
      echo "Successfully compiled ${I2G_MPI_APPLICATION}"
      return 0
    }
    # This function will be called before the MPI executable is finished.
    # A typical case for this is to upload the results to a Storage Elem.
    post_run_hook () {
      echo "Executing post hook."
      echo "Finished the post hook."
      return 0
    }
```

Il *Job Description Language*

Il *JobType* deve essere MPICH anche se il *flavor* scelto è differente.
La variabile *Arguments* deve contenere prima il nome del programma e poi il *flavor* di MPI.
L'*OutputSandbox* deve contenere, oltre al sorgente MPI, il *wrapper* e gli *hook*.
Nell'*hook* di esempio il nome del *file* sorgente coincide con il nome del *file* ".c".

```
     JobType = "MPICH";
     NodeNumber = 8;
     Executable = "mpi-start-wrapper.sh";
     Arguments = "mpi-test OPENMPI";
     StdOutput = "mpi-test.out";
     StdError = "mpi-test.err";
     InputSandbox = {"mpi-start-wrapper.sh","mpi-hooks.sh","mpi-test.c"};
     OutputSandbox = {"mpi-test.err","mpi-test.out"};
     Requirements =
  Member("MPI-START", other.GlueHostApplicationSoftwareRunTimeEnvironment)
&& Member("OPENMPI", other.GlueHostApplicationSoftwareRunTimeEnvironment)
;
```

ESECUZIONE IN LOCALE

Mpi-start può essere facilmente utilizzato anche per l'esecuzione tramite code in locale, ad esempio tramite Torque: è sufficiente fare qualche modifica allo *script* usato in precedenza

```
    #!/bin/bash
    #
    # Pull in the arguments.
    #//// Customized for local execution /////
    WORK_DIR=$WD
    MY_EXECUTABLE=$WORK_DIR/$EXE
```

```
MPI_FLAVOR=$FLAVOR
#//////////////////////////////////////
# Convert flavor to lowercase in order to pass it to mpi-start.
MPI_FLAVOR_LOWER=`echo $MPI_FLAVOR | tr '[:upper:]' '[:lower:]'`
# Pull out the correct paths for the requested flavor.
eval MPI_PATH=`printenv MPI_${MPI_FLAVOR}_PATH`
# Ensure the prefix is correctly set.  Don't rely on the defaults.
eval I2G_${MPI_FLAVOR}_PREFIX=$MPI_PATH
export I2G_${MPI_FLAVOR}_PREFIX
# Touch the executable.It must exist for the shared file-system check.
# If it does not, then mpi-start may try to distribute the executable
# (while it shouldn't do that).
touch $MY_EXECUTABLE
# Setup for mpi-start.
export I2G_MPI_APPLICATION=$MY_EXECUTABLE
export I2G_MPI_APPLICATION_ARGS=
export I2G_MPI_TYPE=$MPI_FLAVOR_LOWER
#//// Customized for local execution /////
export I2G_MPI_PRE_RUN_HOOK=$WORK_DIR/mpi-hooks.sh
export I2G_MPI_POST_RUN_HOOK=$WORK_DIR/mpi-hooks.sh
#//////////////////////////////////////
# If these are set then you will get more debugging information.
export I2G_MPI_START_VERBOSE=1
#export I2G_MPI_START_DEBUG=1
# Invoke mpi-start.
$I2G_MPI_START
```

CONCLUSIONI

La soluzione descritta è stata installata presso il sito INFN-Parma, in cui la *farm* Grid integra i principali *cluster* di calcolo preesistenti come descritto nel documento. La stessa procedura di sottomissione, con piccole modifiche, viene utilizzata dagli utenti per *job* locali o Grid verso le stesse risorse.

La gestione centralizzata consente di automatizzare le installazioni e ottimizzarne la gestione, concentrando nell'unico *Job Manager* le politiche di utilizzo delle risorse.

Uno studio approfondito sull'architettura di *scheduling* deve essere fatto a livello di *middleware* per ottenere una convivenza efficiente tra *job* MPI e non-MPI. Eventuali *cluster* configurati con MPI e dotati di una rete di comunicazione ad alta velocità rischiano di essere sommersi da *job* sequenziali poiché non è prevista in modo nativo la possibilità di riservare risorse a *job* paralleli.