CiTIUS

Centro Singular de Investigación en **Tecnoloxías** da **Información**

Guía de usuario del clúster `ctcomp2`

Diego R. Martínez

v1.0 [Diciembre 2012]

citius.usc.es

Contenidos

Introducción

El clúster `ctcomp2` es un sistema de computación de altas prestaciones (CAP) que proporciona a los usuarios del CITIUS la posibilidad de ejecutar programas computacionalmente exigentes.

En su forma más simple, la utilización de los recursos computacionales de este clúster se puede resumir en los siguientes pasos:

1. Acceder, mediante `ssh`, al clúster. Los ficheros necesarios para la ejecución de los trabajos se pueden importar al espacio de usuario del clúster mediante `scp`.

2. Si utilizamos lenguajes compilados (C, C++, Fortran...) es necesario compilar adecuadamente el código fuente.

3. Escribir un script que indique al gestor de colas los parámetros de la ejecución: el directorio de trabajo, el comando de ejecución, el número de nodos/núcleos necesarios....

4. Enviar, mediante `qsub`, el script al gestor de colas para que lo incluya en la cola de planificación de trabajos, a la espera de ser emitido para su ejecución en los nodos computacionales.

5. La emisión del trabajo a los nodos computacionales se realizará cuando el gestor de colas encuentre un hueco temporal y espacial adecuado para la ejecución del trabajo, en función de los parámetros de ejecución indicados en el script. Es posible utilizar una opción en el script para indicar una dirección de correo electrónico en la que se notificará la finalización de las tareas indicadas en el script.

Descripcion del hardware

El clúster `ctcomp2` está formado por 8 nodos computacionales Blade HP Proliant BL685c G7. Cada nodo dispone de 4 procesadores AMD Opteron 6262 HE (16 núcleos) y 128 GB de RAM. Los nodos están conectados a través de varias redes, con anchos de banda de 1GbE y 10GbE.

El acceso de los usuarios al clúster se realiza a través de una máquina independiente, que denominamos `frontend`. Este `frontend` tiene unas prestaciones limitadas, y únicamente está preparado para gestionar ficheros, compilar código y enviar trabajos al sistema de colas del clúster. No está permitida la ejecución de código paralelo en esta máquina. ¡!

El usuario dispone de los siguientes espacios, dentro del sistema de ficheros del clúster, para ubicar los ficheros relacionados con los trabajos realizados en clúster:

— `/home/local/nome.apelido/`
Este directorio es el `$HOME` del usuario `<nome.apelidos>` y es accesible en todos los nodos del clúster, incluido en el `frontend`. Por defecto, será el directorio de referencia en las ejecuciones de los códigos en los nodos computacionales. La cuota de usuario es limitada, por lo que los usuarios que necesiten gestionar ficheros muy grandes deberán hacerlo a través del directorio `/sfs/`.[1]

— `/sfs/`
Este directorio también está compartido entre todos los nodos de computación y el `frontend`. Este directorio podrá utilizarse como espacio auxiliar durante la ejecución de trabajos, para el almacenamiento de ficheros temporales que deban estar accesibles en todos los nodos o para almacenar los ficheros resultado de una ejecución. En este directorio no se garantiza la conservación permanente de los ficheros que no hayan sido accedidos en los últimos 30 días. Se recomienda utilizar nombres de ficheros/directorios que identifiquen claramente al binomio usuario/programa, para evitar potenciales conflictos entre usuarios.

— `/scratch/`
Cada nodo computacional del clúster dispone de un directorio *scratch* local que puede ser utilizado para almacenamiento temporal local durante la ejecución de una tarea. El contenido de un directorio *scratch* no es visible desde el resto de nodos. El contenido de este directorio no estará accesible desde el "frontend", por lo que no es un lugar adecuado para guardar ficheros con resultados. En cualquier caso, no se garantiza la conservación de ficheros de manera permanente en este directorio. Se recomienda utilizar nombres de ficheros/directorios que identifiquen claramente al binomio usuario/programa, para evitar potenciales conflictos entre usuarios.

[1] En cualquier caso, los usuarios con necesidades de almacenamiento singulares deberán solicitar esos requerimientos a través del sistema de incidencias del CITIUS.

1 Acceso al clúster ctcomp2

Los servicios del clúster son accesibles a través del frontend (ctcomp2.inv.usc.es). El acceso remoto a este servidor se realiza mediante ssh (*secure shell*). Se utiliza el sistema de autentificación centralizado del CITIUS.

ssh
```
local$ ssh [-p<port>] nome.apelido@ctcomp2.inv.usc.es
Password:
ct$
```

Los argumentos de este comando son:

`-p<port>` (*opcional*) Indica el puerto por el cual se conecta el comando scp.

1.1 Autentificación interna

Para el correcto funcionamento del sistema de colas, los usuarios deben poder autentificarse en los nodos computacionales y en el frontend sin que les sea necesario introducir por teclado ningunha información. Como tanto el frontend como los nodos computacionales comparten el $HOME (a través del sistema de ficheros compartido), el usuario debe ejecutar los siguientes comandos la primera vez que accede al frontend (no cubrir nada, simplemente responder con un *Return* a todas las preguntas):

ssh interno
```
ct$ cd $HOME
ct$ ssh-keygen -t rsa
ct$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

1.2 Importación/exportación

El espacio de usuario del clúster es independiente de otros sistemas en el CITIUS, por lo que es necesario importar al espacio de usuario del clúster todos los ficheros necesarios para la ejecución de nuestros programas (por ejemplo, el código fuente o los ficheros de entrada del programa). El comando scp permite el intercambio de ficheros con otros sistemas conectados en red. La sintaxis del comando scp es la siguiente:

scp
```
scp [-P<port>] [-r] <direccion_origen> <direccion_copia>
```

Los argumentos de este comando son:

`-P<port>` (*opcional*) Indica el puerto por el cual se conecta el comando `scp`.

`-r` (*opcional*) Es un argumento que se utiliza cuando `direccion_origen` es un directorio, e indica que se debe copiar de manera recursiva el contenido del directorio.

`<direccion_origen>` Indica la ruta completa del fichero/directorio que se copiará.

`<direccion_copia>` Indica la ruta completa donde queremos ubicar la copia del fichero/directorio.

Ejemplos `scp` en `ctcomp2`

A modo de ejemplo, se muestran varios ejemplos de importación de ficheros. En estos ejemplos, se supone que el puesto de trabajo habitual del usuario `nome.apelido` es `ctXXX.inv.usc.es`. El fichero/directorio que queremos importar está situado en `ctXXX.inv.usc.es`, en el directorio `/datos/work/`, y queremos hacer una copia en el espacio de usuario del clúster en el directorio `work` del `$HOME` del usuario.

Si ejecutamos `scp` en el propio clúster:

scp (@ctcomp2)

```
ct$ scp -P<port> \
  nome.apelido@ctXXX.inv.usc.es:/datos/work/un.fichero \
  $HOME/work/
ct$ scp -P<port> -r \
  nome.apelido@ctXXX.inv.usc.es:/datos/work/directorio/ \
  $HOME/work/
```

Si ejecutamos `scp` en nuestro sistema local (externo al clúster):

scp (@local)

```
local$ scp -P<port>       \
  /datos/work/un.fichero  \
  nome.apelido@ctXXX.inv.usc.es:$HOME/work/
local$ scp -P<port> -r    \
  /datos/work/directorio/ \
  nome.apelido@ctXXX.inv.usc.es:$HOME/work/
```

2 Gestión de software con `modules`

El comando `modules` permite gestionar, de manera eficaz y consistente, múltiples versiones de librerías y sofware para que el usuario utilice la versión adecuada en función de sus requerimientos. Su funcionamiento se basa en el encapsulamiento, dentro de un módulo, de las variables de entorno relacionadas con una versión de software determinada. De este modo, es el propio usuario quien gestiona la utilización de las diferentes versiones de software disponibles en el sistema.

La gestion, a nivel de usuario, de los módulos se realiza con el comando `modules` :

module

```
ct$ module avail
ct$ module list
ct$ module load <module_name>
ct$ module unload <module_name>
ct$ module purge
```

Las opciones son:

`avail` Muestra todos los módulos disponibles en el sistema.

`list` Muestra todos los módulos que están siendo utilizados en la sesión actual.

`load` Activa el módulo `<module_name>`

`unload` Desactiva el módulo `<module_name>`

`purge` Desactiva todos los los módulos de la sesión actual.

El comando `modules` manipula las variables de entorno relacionadas con los *path* del sistema (`PATH`, `LD_LIBRARY_PATH`, etc.), por lo que se recomienda a los usuarios no modificar estas variables de modo arbitrario.

Se recomienda utilizar este comando de manera interactiva. Su uso dentro de `.bashrc` para cargar automáticamente *módulos* habituales no está recomendado, ya que todos los scripts que se ejecuten leen este fichero.

Se recomienda utilizar las versiones por defecto de los difentes módulos. En cualquier caso, el comando `module avail` porporciona una lista completa de todos los los módulos y versiones disponibles.

3 Compilación

Compilación C/C++/Fortran

La colección de compiladores GNU (GNU Compiler Collection, GCC) es accesible en el clúster a través de sus comandos y opciones habituales. Por defecto, los compiladores instalados en el sistema pertenecen a la versión 4.7.2 de GCC (versión por defecto del SO).[2]

GCC

```
ct$ module load gcc
ct$ gcc      -O exemplo.c   -o exemplo
ct$ g++      -O exemplo.cpp -o exemplo
ct$ gfortran -O exemplo.f   -o exemplo
```

Las opciones recomendadas son:

`-O` Genera código optimizado para obtener un mayor rendimiento. Es equivalente a `-O1`. Alternativamente, se pueden utilizar las opciones `-O0`, `-O2` o `-O3`. El número indica el nivel de optimización, siendo 0 el nivel sin ningún tipo de optimización y 3 el nivel con el que se optiene un mayor rendimiento (la opción `-O3` realiza algunas optimizaciones agresivas que pueden generar resultados imprecisos).

`-o <name>` Establece el nombre del fichero ejecutable.

Compilación OpenMP

La colección de compiladores GCC permite la compilación de código OpenMP, indicándolo mediante la opción `-fopenmp`.

GCC + OpenMP

```
ct$ gcc      -O -fopenmp exemplo.c
ct$ g++      -O -fopenmp exemplo.cpp
ct$ gfortran -O -fopenmp exemplo.f
```

[2] Esta versión de los compiladores disponen de una opción de optimización (`-march=bdver1`) para generar código específico para la arquitectura de los nodos del clúster (procesadores Opteron 6200 series, 15th Family *Bulldozer* Interlagos). Esta opción de compilación no garantiza el cumplimiento del estándar matemático definido en GCC, por lo que no se recomienda su uso, salvo en aquellos casos en los que se conozca en profundidad el comportamiento de las opciones de compilación.

Compilación MPI

Para compilar código MPI es necesario cargar un módulo MPI (como, por ejemplo, el módulo `openmpi`), que proporcione los scripts de compilación de código MPI (`mpicc`, `mpicxx`, `mpif77`). Estos scripts hacen llamadas al compilador del lenguaje correspondiente.

GCC+MPI

```
ct$ module load openmpi
ct$ mpicc  -O exemplo.c
ct$ mpicxx -O exemplo.cpp
ct$ mpif77 -O exemplo.f
```

4 Envío de trabajos al sistema de colas Torque/PBS

Los usuarios no pueden conectarse a los nodos computacionales, por lo que la ejecución de trabajos en los nodos computacionales tiene que realizarse, obligatoriamente, a través del sistema de colas Torque/PBS. El sistema de colas registrará por orden cada una de las solicitudes enviadas y las emitirá, para su ejecución en los nodos computacionales, cuando estén disponibles los recursos requeridos.

El envío de trabajos se realiza a través del comando `qsub`, cuyo argumento obligatorio es el nombre de un script de *shell*. El script tiene que disponer de permisos de ejecución. Dentro del script, el usuario debe indicar la acciones que se realizarán en los nodos, una vez que los recursos requeridos estén disponibles (La §4.2: *Ejemplos de scripts* contiene diferentes ejemplos relacionados con los módulos instalados en el clúster).

qsub
```
ct$ chmod u+x <script.sh>
ct$ qsub <script.sh>
```

El comando `qstat` permite al usuario consultar el estado de la cola PBS.

qstat
```
ct$ qstat
ct$ qstat -a
```

El comando `qdel` permite al usuario eliminar un trabajo de la cola PBS, antes de que sea emitido a los nodos computacionales para su ejecución. Este comando necesita como argumento el identificador que PBS le asigna cuando se registra un nuevo trabajo, y que se puede consultar con `qstat`.

qdel
```
ct$ qdel <job_id>
```

4.1 Opciones de configuración Torque/PBS

El sistema de colas PBS permite a los usuarios configurar diferentes aspectos de la ejecución de los trabajos. Las instrucciones de configuración Torque/PBS se indican en los scripts a través de líneas que comienzan (sin espacios) con `#PBS`.[3] También es posible indicar estas opciones directamente como argumentos de `qsub` en la línea de comandos.

[3] De algún modo, estas órdenes son comentarios *especiales* del script, ya que el inicio de los comentarios se indica con el símbolo #.

Algunas de las opciones más comunes son:

`-N` Indica el nombre de referencia de nuestro trabajo en el sistema de colas. Por defecto sería el nombre del ejecutable.
Ej: `#PBS -N myjob`

`-l` Indica los recursos que se solicitan para la ejecución de nuestro trabajo, como el número de núcleos computacionales y el tiempo de ejecución. Los diferentes tipos de recursos se separan por comas.
Ej: `#PBS -l nodes=1:ppn=1,walltime=1:00:00`

- `nodes=`ν`:ppn=`κ: solicitamos ν nodos computacionales, y κ núcleos en cada nodo.[4]
- `walltime=HH:MM:SS`: solicitamos la exclusividad de los recursos durante un tiempo máximo de HH horas, MM minutos y SS segundos. El límite máximo de tiempo permitido es de 168 horas (1 semana).

`-e` Indica el fichero en el que se redireccionará la salida estándar de error de nuestro ejecutable. Por defecto, la salida estándar de error se redirecciona a un fichero con extensión `.eXXX` (donde `XXX` representa el identificador PBS del trabajo).
Ej: `#PBS -e mySTD.err`

`-o` Indica el fichero en el que se redireccionará la salida estándar de nuestro ejecutable. Por defecto, la salida estándar se redirecciona a un fichero con extensión `.oXXX` (donde `XXX` representa el identificador PBS del trabajo).
Ej: `#PBS -o mySTD.out`

`-m` Indica el tipo de eventos que serán notificados por correo electrónico. Los argumentos posibles de esta opción son: `b` cuando el trabajo se emita a los nodos, `a` en caso de que se aborte la ejecución del trabajo inexperadamente y/o `e` cuando el trabajo termine su ejecución sin ningún incidente. Estos argumentos no son excluyentes y se pueden combinar.
Ej: `#PBS -m ae`

[4] No se garantiza la ejecución en exclusividad de los nodos, si no se solicitan los 64 núcleos de un nodo.

`-M` Indica la dirección de correo en la que se notificarán los eventos indicados con la opción `-m`.
Ej: `#PBS -M nombre.usuario@usc.es`

Por defecto, el entorno de ejecución del sistema Torque/PBS define algunas variables de entorno que pueden ser utilizadas dentro de los scripts:

— `$PBS_O_WORKDIR`: contiene el *path* del directorio de trabajo (`$PWD`) desde donde se ha ejecutado el comando `qsub`. Es útil para establer un directorio de referencia durante la ejecución de los trabajos indicados.

4.2 Ejemplos de scripts

A continuación se muestran varios ejemplos de scripts que muestran el conjunto básico de intrucciones para el envio de diferentes tipos de trabajos al gestor de colas.

Ejemplo de trabajo secuencial:

```
#!/bin/bash
#PBS -l nodes=1:ppn=1,walltime=1:00:00
#PBS -N ejemplo
#PBS -e salida.out
#PBS -o salida.err
#PBS -m ae -M <direccion_correo@usc.es>
cd $PBS_O_WORKDIR
./executable
```

Ejemplo de trabajos secuenciales en paralelo:

```
#!/bin/bash
#PBS -l nodes=1:ppn=4,walltime=1:00:00
#PBS -N ejemplo
cd $PBS_O_WORKDIR
(cd dir1; ./executable1) &
(cd dir2; ./executable2) &
(cd dir3; ./executable3) &
(cd dir4; ./executable4) &
wait
```

Ejemplo de trabajo Java:

```
#!/bin/bash
#PBS -l nodes=1:ppn=64,walltime=1:00:00
#PBS -N ej-java
cd $PBS_O_WORKDIR
module load jdk
java executable
```

Ejemplo de trabajo OpenMP:

```
#!/bin/bash
#PBS -l nodes=1:ppn=64,walltime=1:00:00
#PBS -N ej-openmp
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=64
./executable
```

Ejemplo de trabajo MPI:

```
#!/bin/bash
#PBS -l nodes=8:ppn=64,walltime=1:00:00
#PBS -N ej-mpi
cd $PBS_O_WORKDIR
module load openmpi
mpirun -np 512 ./executable
```

Referencia @ctcomp2

CiTIUS
Centro Singular de Investigación en Tecnoloxías da Información

```
# Acceso
ssh -p <port> nome.apelido@ctcomp2.inv.usc.es

# importar
scp -p <port> \
  fich_orixe  \
  nome.apelido@ctcomp2.inv.usc.es:~/fich_destino
scp -p <port> -r \
  dir_orixe      \
  nome.apelido@ctcomp2.inv.usc.es:~/dir_destino

# exportar
scp -p <port>                                  \
  nome.apelido@ctcomp2.inv.usc.es:~/fich_orixe \
  fich_destino
scp -p <port> -r                               \
  nome.apelido@ctcomp2.inv.usc.es:~/dir_orixe  \
  dir_destino
```

```
# Gestión de módulos
module avail
module list
module load <module_name>
module unload <module_name>
module purge
```

```
# Compilación GCC
gcc      -O exemplo.c   -o exe
g++      -O exemplo.cpp -o exe
gfortran -O exemplo.f   -o exe
```

```
# Compilación OpenMP (+GCC)
gcc      -O -fopenmp exemplo.c   -o exe-omp
g++      -O -fopenmp exemplo.cpp -o exe-omp
gfortran -O -fopenmp exemplo.f   -o exe-omp
```

```
# Compilación OpenMPI (+GCC)
module load openmpi
mpicc  -O exemplo.c    -o exe-mpi
mpicxx -O exemplo.cpp  -o exe-mpi
mpif77 -O exemplo.f    -o exe-mpi
```

```
# Manejo de la cola de trabajos
chmod u+x myjob.sh
qsub myjob.sh
qstat
qdel <job_id>
```

```
#!/bin/bash
#
# --- Ejemplo de configuración PBS --- #
#
#PBS -l nodes=1:ppn=1,walltime=1:00:00
#PBS -N proba-PBS
#PBS -e mySTD.err
#PBS -o mySTD.out
#PBS -m ae -M nome.apelido@usc.es
cd $PBS_O_WORKDIR
echo "OK"
```

```
#!/bin/bash
#
# --- Ejemplo trabajo secuencial --- #
#
#PBS -l nodes=1:ppn=4,walltime=1:00:00
#PBS -N ej-secuencial
cd $PBS_O_WORKDIR
(cd dir1; ./exec1) &
(cd dir2; ./exec2) &
(cd dir3; ./exec3) &
(cd dir4; ./exec4) &
wait
```

```
#!/bin/bash
#
# --- Ejemplo trabajo JAVA --- #
#
#PBS -l nodes=1:ppn=64,walltime=1:00:00
#PBS -N ej-java
cd $PBS_O_WORKDIR
module load java
java executable
```

```
#!/bin/bash
#
# --- Ejemplo trabajo OpenMP --- #
#
#PBS -l nodes=1:ppn=64,walltime=1:00:00
#PBS -N ej-openmp
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=64
./executable
```

```
#!/bin/bash
#
# --- Ejemplo trabajo MPI --- #
#
#PBS -l nodes=2:ppn=64,walltime=1:00:00
#PBS -N ej-mpi
cd $PBS_O_WORKDIR
module load openmpi
mpirun -np 128 ./executable
```