Centro Informático Científico de Andalucía
CONSEJERÍA DE LA PRESIDENCIA, INTERIOR, DIÁLOGO SOCIAL Y SIMPLIFICACIÓN ADMIN

POLÍTICA DE USO Y BUENAS PRÁCTICAS DEL CLÚSTER DE SUPERCOMPUTACIÓN PARA USUARIOS

21 de Junio de 2023

Índice de contenido

1. Objetivo

Uno de los principales objetivos del CICA es ayudar a la comunidad científica e investigadora a desarrollar sus trabajos, especialmente cuando estos requieren recursos intensos en computación. Para ello, el CICA pone al servicio de los grupos de investigación y centros de investigación andaluces un clúster de Supercomputación de alto rendimiento.

El presente documento recoge las principales políticas de uso del clúster de Supercomputación del CICA y pretende ser una guía de información y buenas prácticas para los usuarios del servicio.

2. Descripción del servicio

El CICA pone al servicio de los grupos de investigación y centros de investigación andaluces un clúster de Supercomputación (High Perfomance Computing-HPC) formado por más de 50 nodos propios, con más de 500 cores, de los cuales 32 nodos de cálculo (256 cores) están conectados entre si a través de Infiniband DDR a 20 Gbit/seg.

En proceso constante de actualización y mejora de servicio, el acceso al clúster está gestionado con el producto de software libre SGE, que permite organizar en distintas colas los trabajos de los usuarios. Con esta división se consigue que el tiempo de espera para trabajos cortos se vea reducido, evitando que los trabajos largos acaparen el uso del clúster, permitiendo un reparto optimizado de la utilización de los cores que tiene en cuenta la carga de trabajo de cada una de las colas.

Actualmente el clúster cuenta con numerosos programas de cálculo de uso frecuente por los grupos de investigación, estando abierto a la instalación de nuevo software en función de las peticiones de los usuarios.

En abril de 2016 se incorporó a uno de los nodos de cálculo una aceleradora GPU nVidia Tesla K10.

El equipo de Operaciones del CICA es el encargado del mantenimiento e implementación del servicio, así como de ofrecer apoyo técnico a los usuarios que lo requieran. Dentro de este servicio se realizan actividades como la gestión de copias de seguridad, monitorización o la instalación y actualización de software.

3. Guía de uso del clúster

3.1. Requisitos para ser usuario del clúster

Todo investigador que desee usar la infraestructura de Supercomputación del CICA debe pertenecer a un grupo o centro de investigación andaluz.

3.2. Solicitud de alta como usuario del clúster

Para solicitar el alta como usuario del clúster de Supercomputación del CICA, se debe rellenar el siguiente formulario:

https://www.cica.es/servicios/supercomputacion/registro

3.3. Requisitos para acceder al clúster

Para acceder al clúster necesita:

- PC con conexión a Intenet y herramienta para conectar por protocolo SSH. (Ej. terminal de Linux, Putty en Windows).

- Usuario y contraseña (enviado a todo usuario registrado a través del formulario descrito en el punto 3.2).

3.4. Primeros pasos en el clúster

El usuario debe acceder al clúster a partir de una conexión SSH al servidor *sesamo.cica.es* con sus credenciales (usuario y contraseña):

Ej.Linux: ssh usuario@sesamo.cica.es

Una vez dentro del sistema, puede:

- Subir ficheros para trabajar en el clúster, almacenándolos en su directorio personal (/home/usuario/).

- Ir por SSH al servidor de desarrollo y compilación de programas *devel.hpc.cica.es* para compilar sus propios programas.

- Ir por SSH al servidor de gestión de colas *pool.hpc.cica.es* para lanzar nuevos trabajos y revisar el estado en que se encuentren los trabajos que haya lanzado.

3.5. Directorio personal

Los servidores del clúster comparten por red el /home del usuario de tal forma

que desde cualquier servidor podrá acceder a /home/usuario/. Por ejemplo, los resultados de una compilación en *devel* automáticamente estarán también en *pool*.

Cada usuario dispone de un espacio máximo de 300 GB para almacenar los ficheros necesarios para su investigación, no pudiendo almacenar mayor cantidad de información superado este umbral.

3.6. Sistema de colas

Lo más importante para lanzar trabajos en el clúster es saber qué colas hay en *pool* y para qué se usan:

- simple: Para lanzar trabajos que no necesitan más de un procesador y menos de 4 GB de RAM.
- multicore: Para lanzar trabajos que requieran más de un procesador y/o más de 4 GB de RAM.
- mpi: Para lanzar trabajos que utilicen librerías mpi.
- cuda: Para lanzar trabajos que utilicen GPU.

Los tres primeros contenedores se subdividen en 4 colas específicas ordenadas por duración: diaria (estimación de trabajos de un día como máximo), corta (duración máxima de 3 días), media (duración máxima de 1 semana), larga (duración máxima de 1 mes).

De esta forma, los nombres de las colas habilitadas son, según su propósito:

- Programas que NO necesitan multihilo ni proceso distribuido, y no necesitan más de 4 GB:
  diaria_simple, corta_simple, media_simple, larga_simple.
- Programas multihilo o que necesiten más de 4 GB de RAM:
  diaria_multicore, corta_multicore, media_multicore, larga_multicore.
- Programas de proceso distribuido que usen MPI:
  diaria_mpi, corta_mpi, media_mpi, larga_mpi.
- Programas que utilicen GPU:
  cuda.

Para trabajos de especificaciones distintas a las señaladas, debe contactar con el equipo a través de soporte@cica.es

3.7. Creación de fichero para el sistema de colas

Para lanzar cualquier trabajo al clúster hay que crear un pequeño script donde configurar los parámetros necesarios para el entorno y las colas.

En el Anexo I tiene un completo tutorial sobre cómo escribir dichos scripts. No deje de leerlo para trabajar con SGE.

3.8. Uso de GPU

En abril de 2016 se incorporó a uno de los nodos de cálculo una aceleradora GPU nVidia Tesla K10 con las siguientes características:

- 2 GPUs GK-104 con microarquitectura Kepler.
- 1536 cores cada GPU (3072 en total) a 745 MHz
- 3584 MB de RAM GDDR5 a 2,5GHz y bus de 256 bits.
- 4500GFlops en single precision. 190GFlops en double precision.

El uso de este hardware por parte de las aplicaciones necesita de compilaciones especiales y depende de cada aplicación. Aquellos usuarios que quieran ejecutar aplicaciones de su interés sobre esta GPU deben ponerse en contacto con nosotros a través de soporte@cica.es indicándonos qué aplicación desea usar.

En las pruebas de uso se comparó el rendimiento de dos aplicaciones con y sin CUDA. Como se puede observar en la siguiente tabla, el incremento de rendimiento puede llegar a ser espectacular:

Aplicación	Fichero de entrada	Tiempo de ejecución sin CUDA	Tiempo de ejecución con CUDA
LAMMPS (MPI en 8 cores)	bench/FERMI/in.eam	77,29 seg.	17,87 seg.
BEAST (BEAGLE)	examples/Benchmarks/benchmark2.xml	25,12 min.	6,13 min.

3.9. Monitorización de trabajos

En el sistema de colas puede ver el estado de sus trabajos, monitorizar el comportamiento del nodo que está ejecutando sus trabajos o cancelarlos.

- Información detallada sobre un trabajo. Usar el comando: qstat -j <número de trabajo>
- Cancelar trabajos. Buscar el número del trabajo mediante el comando: qstat -u <username>

  y cancelarlo con el comando: qdel <número de trabajo>

4. Política de uso

- Todo usuario general del clúster debe pertenecer a un grupo o centro de investigación andaluz. Deberá proporcionar una cuenta de la universidad o centro al que pertenece y mantener el contacto desde la misma. Una vez creada una cuenta de usuario, se establecerá una validez de 12 meses renovables siempre que haga uso del servicio.

- Las cuentas con inactividad superior a 10 meses, causarán baja de la infraestructura del clúster y se eliminará la información que pudiera contener en su directorio personal.

- Cada usuario dispone de un espacio máximo de 300 GB para almacenar los ficheros necesarios para su investigación, no pudiendo almacenar mayor cantidad de información superado este umbral.

- Para crear cuentas a alumnos universitarios, el usuario del clúster como investigador principal y responsable deberá autorizar la solicitud del mismo. Estas cuentas tendrán validez por 4 meses.

- El uso del clúster se debe ceñir a:

    - Almacenar en /home/usuario los ficheros estrictamente necesarios para ejecutar sus cálculos.

    - Una vez obtenidos los resultados, descargárselos en su equipo personal liberando el espacio de /home/usuario.

- El usuario debe procurar usar la cola de menor duración posible para sus trabajos.

- Los usuarios pueden instalar sus propios programas en su directorio HOME. Para ello deben usar la máquina devel como plataforma de desarrollo y compilación. Si no saben como, pueden solicitar ayuda al equipo técnico.

- En caso de necesitar usar software comercial, el usuario debe aportar una licencia válida para su instalación y uso en el clúster.

- Si el usuario necesita que se realice copia de seguridad de algunos de sus ficheros, debe almacenarlos en la carpeta "backup" de su directorio HOME. El resto de carpetas no son salvaguardadas, por lo que cualquier información eliminada de éstas no podrá ser recuperada.

Para cualquier duda o asistencia, puede contactar con el equipo técnico a través de https://www.cica.es/el-cica/contacto/ o escribiendo a soporte@cica.es.

Anexo I. Uso del sistema de gestión de recursos SGE

En el clúster del CICA está instalado el programa Sun Grid Engine (SGE) en todos los ordenadores (nodos) que lo componen. El objetivo de SGE es ajustar los recursos existentes en el clúster (procesadores, memoria, tiempo de ejecución) a los trabajos de cálculo que envíen los usuarios.

Un 'trabajo' para SGE es simplemente un shell script escrito por el usuario. Este shell script ejecuta cualquier número de pasos para llevar a cabo la tarea deseada: copiar o mover ficheros/carpetas de un sitio a otro, borrarlos, cambiarse de una carpeta de trabajo a otra dentro del espacio donde el usuario tiene permiso para hacerlo, ejecutar programas que hacen cálculos, etc. También en ese script el usuario puede detallar qué requisitos necesita cumplir el nodo que vaya a ejecutar su trabajo: cuántos procesadores o memoria necesita su trabajo para funcionar.

Cuando el usuario ha terminado de escribir su script con los pasos para llevar a cabo el trabajo que le interesa, simplemente entrega el script a SGE para su ejecución. SGE se encargará de encontrar un nodo en el clúster que cumpla los requisitos indicados y lanzará en él el trabajo programado.

El mecanismo básico que SGE usa para organizar los trabajos y saber en qué máquinas lanzarlos es el sistema de colas. Una 'cola' es la agrupación bajo un mismo nombre de varios nodos de cálculo que comparten alguna característica en común. Esos nombres pueden ser usados a la hora de entregar un trabajo a SGE para indicarle dónde quiere el usuario ejecutar su programa. En el clúster del CICA hay varias colas distintas dependiendo del número de cores y memoria que tienen las máquinas que la atienden y del tiempo máximo que un proceso puede estar en ejecución en el clúster.

En resumen, el proceso de usar los recursos de un clúster con SGE es:

- El usuario escribe un fichero .sge en el que especifica mediante un shell script los pasos para ejecutar su cálculo y los requisitos necesarios.
- Ese script se entrega a SGE. A partir de aquí el usuario puede desconectarse del clúster o dedicarse a preparar otro fichero .sge porque ya no es necesaria más intervención por su parte.
- SGE procesa los requisitos indicados en el script (entre ellos, en qué cola quiere el usuario ejecutar su programa) y busca en la cola indicada un nodo que los cumpla.
- Si las máquinas que podrían hacer ese trabajo están ocupadas, SGE deja el trabajo en la cola hasta que haya recursos libres para ejecutarlo.
- Una vez que SGE descubre que ya se puede lanzar un trabajo, se encarga de entregar al nodo elegido el script y esa máquina lo ejecuta.
- El resultado de la ejecución del script son los ficheros de salida produci-

dos por la ejecución de los programas que interesan al usuario más otros ficheros donde se guardan los mensajes y errores que habrían salido por pantalla si el script se hubiese ejecutado de forma interactiva.

1. Tipos de trabajos que se pueden ejecutar en SGE

Los programas que los usuarios utilizan para sus cálculos están desarrollados con diferentes técnicas y, por tanto, SGE puede ejecutar trabajos de diferentes formas:

Trabajos simples

Un programa que utiliza un solo procesador en un único nodo. Un ejemplo de un trabajo simple sería un programa que toma un fichero que contiene una imagen, la procesa siguiendo algún algoritmo y genera una copia de esa imagen añadiéndole color. Si el programa no está construido con ningún esquema de paralelismo, usará un procesador en una máquina.

Trabajos paralelos multiprocesador (o de memoria compartida)

Un programa que se ejecuta en un solo nodo, pero usa varios procesadores de esa máquina. Este tipo de programas pueden estar construidos usando la especificación OpenMP o la POSIX Threads. Según esté hecho el programa con uno u otro esquema de paralelismo la forma de lanzar este tipo de trabajos es ligeramente distinta, como veremos. Siguiendo con el ejemplo anterior, si el programa fuese capaz de usar varios procesadores de una máquina para procesar la imagen, estaríamos ante un trabajo paralelo de memoria compartida.

Trabajos paralelos distribuidos

Este tipo de programas están diseñados para funcionar de forma paralela pero distribuyen su carga de cálculo entre varios nodos simultáneamente. Usan uno o varios procesadores en varios ordenadores distintos a la vez. Este tipo de programas están construidos usando la especificación MPI (Message Passing Interface). En el clúster del CICA hay dos implementaciones de MPI: OpenMPI y MPICH2. Un ejemplo podría ser un programa que aplica diferentes algoritmos para procesar unos datos, compara los resultados de acuerdo a algún criterio y genera como salida los mejores resultados. Si ese programa usa procesadores en distintas máquinas para aplicar los diferentes algoritmos a los datos, estaríamos ante un programa paralelo distribuido.

Array Jobs

Es una forma de lanzar la ejecución simultánea de múltiples copias del mismo programa pero cada una de ellas procesando diferentes datos. Un caso típico de uso sería el del programa simple (no paralelo) comentado más arriba: supongamos que un usuario tiene que procesar 5.000 imágenes. Puede crear un array job que reparta las 5.000 ejecuciones de su programa entre todos los no-

dos disponibles en el cluster asignando a cada copia del programa una de las imágenes para su procesamiento. Cada tarea individual no es paralela pero, sin embargo, el hecho de poder repartir el trabajo entre muchas máquinas a la vez es una forma de hacer procesamiento paralelo. El array job es la manera que tiene SGE de permitir al usuario expresar este tipo de tareas con poco esfuerzo.

2. Opciones para indicar requisitos de los trabajos

-cwd	Define el directorio desde el que se lanza el trabajo como directorio de trabajo. Las rutas de los ficheros serán relativas a este.
-wd <*ruta*>	Define <*ruta*> como directorio de trabajo. Las rutas de los ficheros serán relativas a <*ruta*>.
-N <*nombre*>	Nombre que el usuario da a su trabajo
-S */bin/tcsh* | */bin/bash*	Shell que debe usarse para interpretar los comandos del script.
-o <*fichero*>	Direcciona la salida estándar al <*fichero*> indicado
-e <*fichero*>	Direcciona la salida de mensajes de error a <*fichero*>
-j y	Indica a SGE que tanto la salida estándar (opcion -o) como el error estándar (opción -e) deben guardarse juntos en el fichero indicado por la opción -o. Por tanto, no es necesaria la opción -e.
-q <*cola*>	Cola a la que se envía el trabajo
-l h_vmem=<*X*>G	Memoria libre en gigabytes que debe tener el nodo donde se vaya a ejecutar este trabajo. Este parámetro es opcional. Si no se indica, SGE asumirá que el programa necesita 2GB de memoria. Esta opción indica la memoria necesaria por slot (vea más abajo *-l slots*).
-l slots=<*n*>	Número de slots (procesadores) que este trabajo necesita en el nodo donde se vaya a ejecutar. Este parámetro es opcional. Si no se indica, SGE asumirá que el trabajo ocupa un slot.
-t <*numero_inicial*>-<*numero_final*>	Indica que este trabajo es un array job y que se compone de tareas que irán numeradas desde *numero_inicial* hasta *numero_final.*
-pe <*openmpi* | *smp*> <*núm. hilos*>	Informa a SGE que este trabajo ejecuta un programa paralelo (ya sea de memoria compartida o distribuido) y que usará *n* hilos de ejecución. Según el nombre del entorno paralelo que usemos, SGE sabrá si debe prepararse para ejecutar un trabajo MPI (entorno openmpi) o un trabajo de memoria compartida (entorno smp). Si se usa esta opción no debe usarse la opción *-l slots.*

3. Uso de la partición /scratch

De forma general se debe usar la partición /scratch como lugar donde colocar o generar ficheros durante sus cálculos. Les recordamos que el $HOME del usuario se monta a través de red en todos los nodos, por lo que el acceso a datos es más lento que la propia partición de /scratch, que es local para cada nodo.

Entonces, el proceso a seguir por cada usuario sería:

- copiar los ficheros que necesite al /scratch de los nodos,
- crear allí los ficheros de salida,
- al final de la ejecución del script, copiar los ficheros que necesite a su $HOME para que vuelva a tenerlos accesibles en toda la red..
- borrar los ficheros de la partición de /scratch.

Vamos a ver un ejemplo más elaborado, que hace uso de la partición /scratch:

Supongamos que quiero ejecutar un programa escrito en lenguaje R. En la máquina *devel* editamos un fichero que llamaremos R-prueba.sge con el siguiente contenido:

```
#$ -S /bin/bash
#$ -N R-prueba
#$ -wd /home/heisenberg
#$ -o R-prueba.salida
#$ -e R-prueba.err
#$ -q diaria_multicore
#$ -pe smp 2-8
#$ -l h_vmem=8G
#
# Copio el fichero de entrada a un subdirectorio mio en /scratch
mkdir -p /scratch/heisenberg/R-prueba
cp R-benchmark-25.R /scratch/heisenberg/R-prueba
# Ahora que he preparado el entorno de trabajo
# en el nodo en el que se va a ejecutar mi programa, lo lanzo
#
export OMP_NUM_THREADS=$NSLOTS
module load R-3.6.1
R --slave --no-save < /scratch/heisenberg/R-prueba/R-benchmark-25.R
#
# Limpio el scratch
# Si el proceso hubiese dejado ficheros de salida que me interesan
# los copio antes a mi /home:
#  cp  /scratch/heisenberg/R-prueba/algun-fichero-de-salida  /home/heisenberg/...
rm -rf /scratch/heisenberg/R-prueba
```

Se lo entregamos al sistema de colas del clúster con el comando:

qsub R-prueba.sge

Una vez lanzado, podemos ver su estado con:

qstat

que nos mostrará el estado de todos los trabajos del usuario 'heisenberg'.

La salida del comando anterior será algo así:

```
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
    513 0,000 R-prueba   heisenberg       qw    09/30/2013 16:06:53                                    1
```

La columna *state* con valor *qw* nos dice que el trabajo está esperando que se encuentre un nodo de cálculo libre para ejecutar el trabajo.

Cuando el sistema de colas encuentra una máquina libre y le envía el trabajo para su ejecución, la salida de qstat será:

```
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
513 0,555 R-prueba   heisenberg       r     09/30/2013 16:07:06 diaria_multicore@nova31.hpc.ci   1
```

Observe cómo ahora el estado es r (running) y se indica la *cola@nodo* en el que se está ejecutando el programa.

Sin embargo, habrá veces que el trabajo de mover ficheros de un directorio a otro no compense. Este será el caso de ficheros de entrada o salida pequeños o, aún siendo grandes, que sólo se use una pequeña parte del mismo. En ese caso, es posible que sea más rápido dejarlos en nuestro /home y acceder a ellos directamente, sin perder tiempo en copiarlos antes al nodo de ejecución.

4. Ejemplos de diferentes tipos de trabajos

Veamos ahora varios ejemplos de scripts .sge que permitirán al usuario comprender mejor cómo se junta todo lo explicado hasta ahora. Tenga en cuenta que los scripts .sge siempre se deben escribir y lanzar desde la máquina *devel*.

Ejemplo 1: Un trabajo trivial.

Supongamos que el usuario, usando un editor de texto como vim o nano escribe este fichero prueba-trivial.sge:

```
# Shell que voy a usar
#$ -S /bin/bash
```

```
# Nombre de la tarea. Algo descriptivo para el usuario
#$ -N prueba-trivial
# Directorio de trabajo. Los ficheros que siguen son relativos a este direc-
torio
#$ -wd /home/heisenberg
# Fichero donde se guardará la salida de ejecución del programa. Lo que
saldría por pantalla, vamos
#$ -o prueba-trivial.salida
# Fichero donde se guardaran errores de ejecucion que se produzcan
#$ -e prueba-trivial.err
# Cola a la que se envía el trabajo
#$ -q diaria_simple
#
# Esto es lo que hace el trabajo y se ejecuta en el nodo que SGE elija
maquina=`hostname`
echo "Hola, mundo!"
echo "Desde el nodo: $maquina"
```

Para ejecutarlo, el usuario sólo tendría que entregárselo a SGE así:

qsub prueba-trivial.sge

Si quiere ejecutarlo usted, simplemente cambie la línea donde está la opción -wd para ajustarla a su directorio de inicio.

Ejemplo 2: Enviando un trabajo simple.

Este ejemplo muestra el script .sge que sería necesario para ejecutar un trabajo simple como el referido antes que procesa una imagen. No es un proceso paralelo, simplemente se ejecuta en un procesador de un nodo. Supongamos este fichero llamado procesa-foto.sge:

```
# Shell que voy a usar
#$ -S /bin/bash
# Nombre de la tarea. Algo descriptivo para el usuario
#$ -N color-imagen
# Directorio de trabajo. Los ficheros que siguen son relativos a este direc-
torio
#$ -wd /home/heisenberg/almacen-fotos
# Fichero donde se guardará la salida de ejecución del programa. Lo que
saldría por pantalla, vamos
#$ -o color-imagen.salida
# Fichero donde se guardaran errores de ejecucion que se produzcan
#$ -e color-imagen.err
# Cola a la que se envía el trabajo
#$ -q diaria_simple
#
```

```
# Esto es lo que hace el trabajo. Se ejecuta en el nodo que SGE elija
/home/heisenberg/bin/procesa-imagen --input foto-001.png --output foto-procesada-001.pgn
```

Como ve, este trabajo lo que hace es ejecutar un programa llamado 'procesa-imagen' al que se le pasan sus opciones correspondientes para que haga su tarea.

Como un fichero .sge es un script, sería posible modificarlo para pasar el nombre de la imagen a procesar como un parámetro en la línea de comando. Si modificamos la línea */home/heisenberg/bin/procesa-imagen...* y la escribimos así: /home/heisenberg/bin/procesa-imagen --input $1 --output $2

También se podría poner el trabajo en la cola para que maneje distintas fotos:

```
qsub procesa-foto.sge foto-001.png foto-procesada-001.png
qsub procesa-foto.sge foto-002.png foto-procesada-002.png
...
```

Ejemplo 3: Un trabajo paralelo de memoria compartida (OpenMP).

Supongamos que nuestro usuario de ejemplo quiere ejecutar un programa paralelo que está construido usando OpenMP. Él sabe que su programa para funcionar bien necesita máquinas de 4 procesadores y, al menos, 8GB de memoria. Este sería su script .sge:

```
# Shell que voy a usar
#$ -S /bin/bash
# Nombre de la tarea. Algo que sea descriptivo para el usuario
#$ -N find-blue-meth
# Directorio de trabajo. Los ficheros que siguen son relativos a este directorio
#$ -wd /home/heisenberg/experiments
# Fichero donde se guardará la salida de ejecución del programa. Lo que saldría por pantalla, vamos
#$ -o resultado.salida
# Los errores van al mismo fichero indicado en la opción -o
#$ -j y
# Cola a la que se envía el trabajo
#$ -q diaria_multicore
# Sólo ejecutar en nodos que tengan al menos 8GB de memoria por slot
#$ -l h_vmem=8G
# Se usa el entorno paralelo 'smp' y en máquinas con al menos 4 cores
#$ -pe smp 4
#
# Esto es lo que hace el trabajo y se ejecuta en el nodo que SGE elija
```

```
export OMP_NUM_THREADS=$NSLOTS
/home/heisenberg/bin/prog-omp estado-inicial.data --output resultados.da-
ta
```

En este script hay un par de puntos importantes:

- Si va a ejecutar programas que usen varios procesadores debe usar las colas *_multicore porque esas colas están atendidas por las máquinas con más cores. Por el contrario, las colas *_simple están atendidas por máquinas con pocos procesadores.
- La variable de entorno OMP_NUM_THREADS tiene significado para un programa construido con OpenMP: le indica cuántos hilos de ejecución tiene que usar en la máquina. SGE predefine, en tiempo de ejecución, la variable NSLOTS al número de slots disponibles para la tarea en el nodo de ejecución concreto que se vaya a usar.
- Este trabajo pide 32GB de memoria en el nodo que lo vaya a ejecutar: 4 slots x 8GB.

El hecho de que SGE defina en la variable NSLOTS cuántos procesadores hay asignados en un nodo concreto, permite una sintaxis alternativa en la opción -pe:

```
-pe smp 2-8
export OMP_NUM_THREADS=$NSLOTS
/home/heisenberg/bin/prog-omp estado-inicial.data --output resultados.da-
ta
```

Esto indica a SGE que reserve, dependiendo de lo que haya disponible en el nodo, entre 2 y 8 procesadores. Y de ese número de procesadores disponibles es informado el programa que se va a ejecutar a través de la asignación `export OMP_NUM_THREADS=$NSLOTS`.

Ejemplo 4. Un trabajo paralelo de memoria compartida (POSIX Threads).

Este tipo de programas paralelos se manejan en SGE de forma muy parecida a los del caso anterior. Vamos a ver este ejemplo, que es una pequeña variación del anterior script:

```
# Shell que voy a usar
#$ -S /bin/bash
# Nombre de la tarea. Algo que sea descriptivo para el usuario
#$ -N find-blue-meth
# Directorio de trabajo. Los ficheros que siguen son relativos a este direc-
```

```
torio
#$ -wd /home/heisenberg/experiments
# Fichero donde se guardará la salida de ejecución del programa. Lo que saldría por pantalla, vamos
#$ -o resultado.salida
# Los errores van al mismo fichero indicado en la opción -o
#$ -j y
# Cola a la que se envía el trabajo
#$ -q diaria_multicore
# Sólo ejecutar en nodos que tengan al menos 8GB de memoria para cada slot
#$ -l h_vmem=8G
# Se usa el entorno paralelo 'smp' y en máquinas con entre 2 y 8 cores
#$ -pe smp 2-8
#
# Esto es lo que hace el trabajo y se ejecuta en el nodo que SGE elija

/home/heisenberg/bin/prog-thread --num-procs=$NSLOTS estado-inicial.-data --output resultados.data
```

Observe que este trabajo pide entre 16 y 64GB de memoria en el nodo de ejecución dependiendo del número de slots asignados.

Normalmente este tipo de programas tienen algún parámetro que permite al usuario indicar cuántos hilos deben ejecutarse. En este caso, hemos supuesto que *prog-thread* admite una opción *--num-procs* para que el usuario indique cuantos hilos deben arrancarse. Normalmente hay que mirar la documentación del programa para saber como indicarlo.

Ejemplo 5. Un trabajo paralelo de memoria distribuida (OpenMPI).

Vistos los ejemplos anteriores este caso es fácil de entender, aunque hay varios puntos importantes que merece la pena destacar. Veamos primero el ejemplo:

```
# Shell que voy a usar
#$ -S /bin/bash
# Nombre de la tarea. Algo que sea descriptivo para el usuario
#$ -N find-blue-meth
# Directorio de trabajo. Los ficheros que siguen son relativos a este directorio
#$ -wd /home/heisenberg/experiments
# Fichero donde se guardará la salida de ejecución del programa. Lo que saldría por pantalla, vamos
#$ -o resultado.salida
# Los errores van al mismo fichero indicado en la opción -o
```

```
#$ -j y
# Cola a la que se envía el trabajo
#$ -q diaria_mpi
# Se usa el entorno paralelo 'openmpi' y con 32 hilos de ejecucion
#$ -pe openmpi 32
#
# Esto es lo que hace el trabajo y se ejecuta en los nodos que SGE elija.
# La información sobre los nodos entre los que se va a distribuir el cálculo
# la pasa SGE automáticamente al entorno OpenMPI

module load mpi/openmpi-x86_64

mpiexec --mca btl openib,self,sm --mca plm_rsh_agent rsh --np 32 /home/
heisenberg/bin/prog-MPI estado-inicial.data --output resultados.data
```

Puntos importantes:

- El entorno paralelo (parámetro -pe) ahora es *openmpi* seguido del número de procesos que se van a distribuir entre los diferentes nodos.
- Los trabajos con MPI deben ser enviados a colas *_mpi porque es donde están las máquinas más adecuadas para tratar este tipo de programas. Esas colas están servidas por máquinas con interfaces Infiniband que permiten la comunicación muy rápida entre los diferentes hilos de ejecución que se distribuyen entre las diferentes máquinas.
- El comando mpiexec mostrado (mpiexec --mca btl openib,self,sm --mca plm_rsh_agent rsh --np 32) debe ser puesto igual en los ficheros .sge que haga el usuario para asegurarse que su programa MPI funciona bien.
- La opción *--np* del comando mpiexec debe usar el mismo número que se ha indicado en la opción *-pe openmpi*
- El usuario que vaya a usar programas con OpenMPI, debe de añadir en su script SGE antes de la ejecución del comando mpirun/mpiexec la siguiente línea: module load mpi/openmpi-x86_64.

Ejemplo 6. Un array job.

Recuperamos el caso de ejemplo que explicamos antes: Un usuario tiene que usar un programa para procesar 5.000 ficheros de datos. En vez de lanzar 5.000 órdenes qsub para procesar cada uno de sus ficheros, el usuario puede crear un array job. Vamos a suponer que los nombres de sus ficheros de datos siguen este esquema: 1-fichero.input, 2-fichero.input... 5000-fichero.input. Con este esquema en los nombres de los ficheros que hay que procesar, es muy fácil hacer un array job que los procese:

```
# Shell que voy a usar
```

```
#$ -S /bin/bash
# Nombre de la tarea. Algo que sea descriptivo para el usuario
#$ -N experimento-21
# Directorio de trabajo. Los ficheros que siguen son relativos a este direc-
torio
#$ -wd /home/heisenberg/experimento21
# Fichero donde se guardará la salida de ejecución del programa. Lo que
saldría por pantalla, vamos
#$ -o fichero-$SGE_TASK_ID.salida
# Fichero donde se guardaran errores de ejecución que se produzcan
#$ -e fichero-$SGE_TASK_ID.err
# Cola a la que se envía el trabajo
#$ -q diaria_simple
# Declaracion de un array job de 5000 trabajos
#$ -t 1-5000
#
# Esto es lo que hace el trabajo y se ejecuta en el nodo que SGE elija
/home/heisenberg/bin/procesa-fichero --input $HOME/almacen-datos/
$SGE_TASK_ID-fichero.input --output $SGE_TASK_ID-fichero.data
```

El parámetro *-t* es el que indica que este script define un array job de 5000 trabajos. SGE va colocando en la variable SGE_TASK_ID un número entre 1 y 5000 y a continuación ejecuta el el script, sustituyendo cada $SGE_TASK_ID por su valor. Para ejecutar las 5000 tareas usa todos los nodos disponibles en la cola *diaria_simple*.

El concepto más básico que hay que tener en cuenta para sacar provecho a un array job es que las diferentes ejecuciones de nuestro programa deben poder distinguirse unas de otras en algo que podamos construir a partir del número que viene dado en SGE_TASK_ID. De esa forma podemos dar diferentes parámetros de funcionamiento a nuestro programa. En el ejemplo, nos basamos en los valores que va tomando SGE_TASK_ID para construir a partir de él el nombre del fichero de entrada y el nombre del fichero de salida que cada copia de *procesa-fichero* tiene que tratar.

Ejemplo 7. Alto rendimiento: un array job de programas de memoria compartida.

Vamos a ver un último ejemplo que use todos los elementos vistos hasta ahora: vamos a suponer que el usuario quiere lanzar la ejecución de un array job y además cada tarea individual consiste en ejecutar un programa OpenMP que necesita 4 procesadores y 32GB de memoria para funcionar bien. Este sería el script .sge:

```
#$ -S /bin/bash
```

```
#$ -N experimento-22
#$ -wd /home/heisenberg/experimento22
#$ -o fichero-$SGE_TASK_ID.salida
#$ -e fichero-$SGE_TASK_ID.err
#$ -q diaria_multicore
#$ -t 1-300
#$ -l h_vmem=8G
#$ -pe smp 4
#
export OMP_NUM_THREADS=$NSLOTS
/home/heisenberg/bin/procesa-paralelo --input $HOME/almacen-datos/$SGE_TASK_ID-fichero.input --output $SGE_TASK_ID-fichero.dat
```