Acceso al cluster multicore

Características del cluster

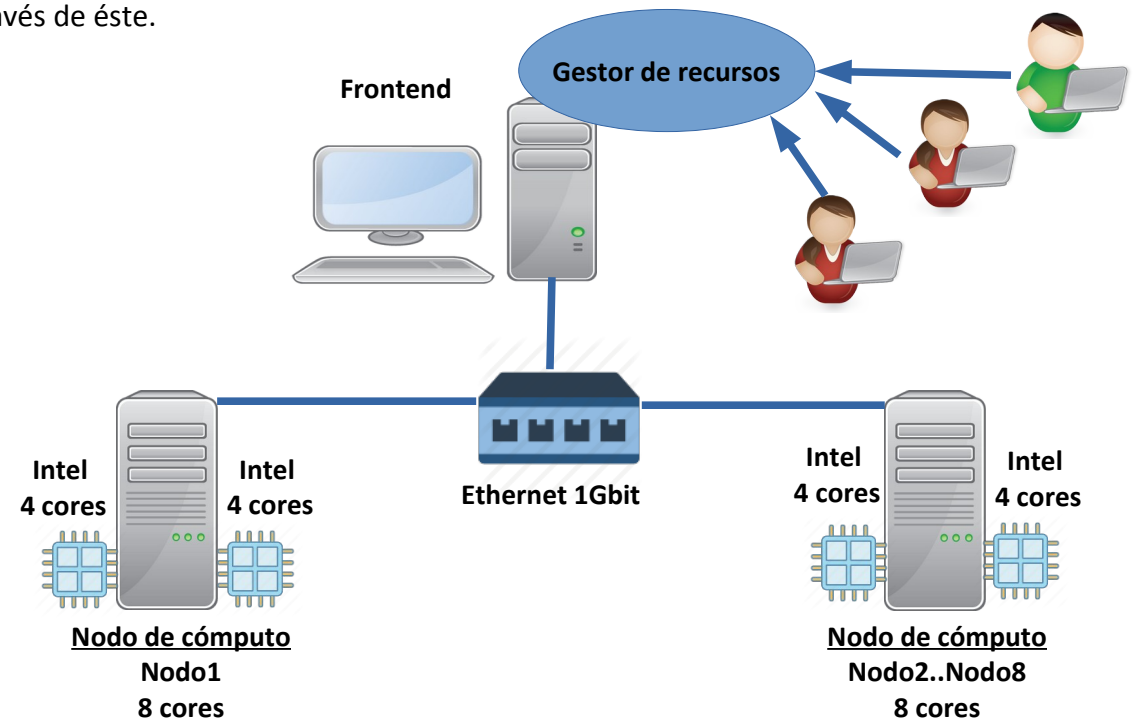
La cátedra posee un cluster multicore para desarrollo compuesto por un frontend y **ocho (8)** nodos de procesamiento.

El frontend es el equipo que centraliza los servicios y desde donde se envían a ejercutar aplicaciones paralelas a los nodos de procesamiento.

Los nodos de procesamiento son los equipos que realizan el cómputo de la aplicación paralela. Cada nodo posee 8GB de memoria RAM y dos procesadores Intel Xeon E5405 a 2.0GHz de 4 cores cada uno.

Todos los equipos están conectados a 1Gbit Ethernet.

El cluster es un recurso compartido por varios usuarios. Cada usuario debería realizar las ejecuciones sobre el mismo de forma exclusiva, es decir sin que la ejecución de otro usuario interfiera. Para evitar conflictos entre las distintas ejecuciones de usuario, el cluster posee un gestor de recursos. Cada usuario deberá ejecutar sus aplicaciones a través de éste.



Acceso remoto al cluster

El cluster se accede de forma remota a través del protocolo SSH.

SSH es un protocolo de red que permite el acceso seguro a un equipo remoto desde una consola en modo texto.

Los pasos que deberá seguir cada usuario para ejecutar una aplicación paralela sobre el cluster son:

1. Transferir el código fuente de la aplicación desde su máquina al cluster
2. Acceder a la consola de texto y compilar
3. Ejecutar la aplicación a través del gestor de recursos
4. Recuperar los resultados

1) Transferir el código fuente de la aplicación desde su máquina al cluster

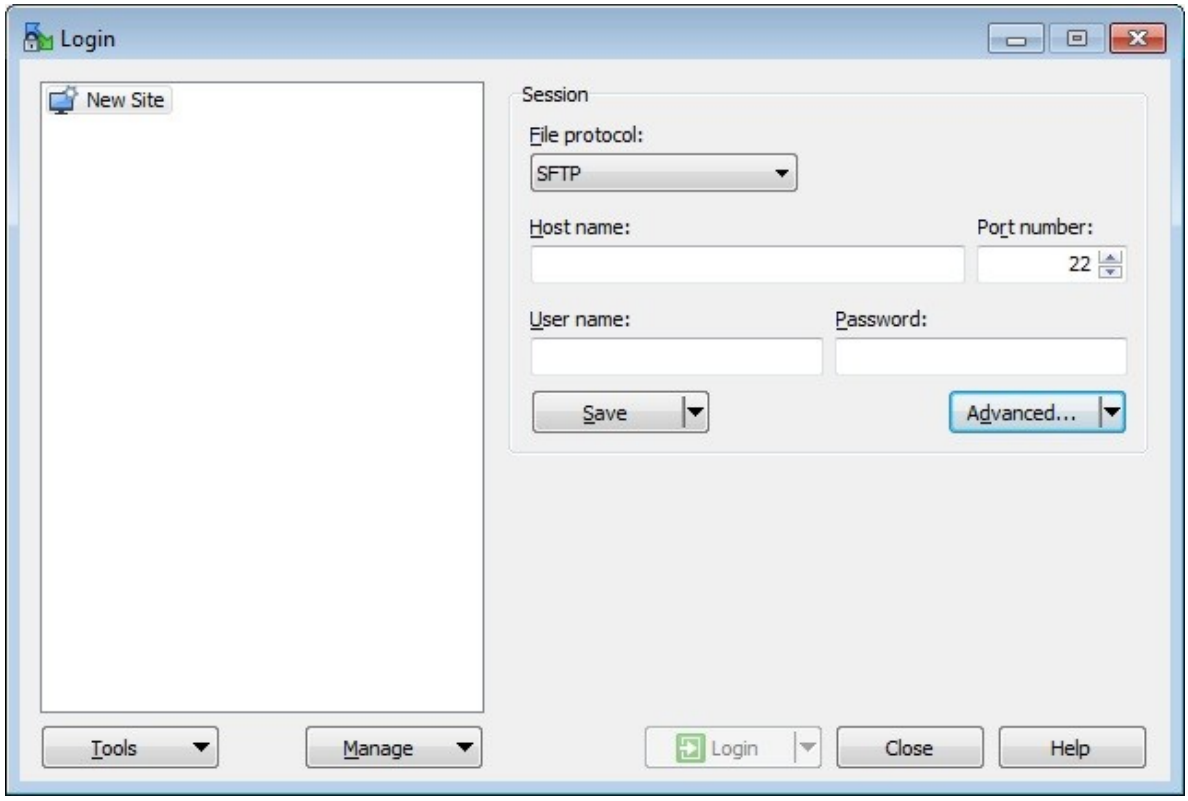
La transferencia del código fuente de la aplicación hacia el cluster se realiza mediante el protocolo SCP, un protocolo de red basado en SSH diseñado para transferencias de archivos seguras.

La forma de usar SCP varía de un sistema operativo a otro, ya sea Windows o Linux. A su vez, varía entre las distintas distribuciones de Linux

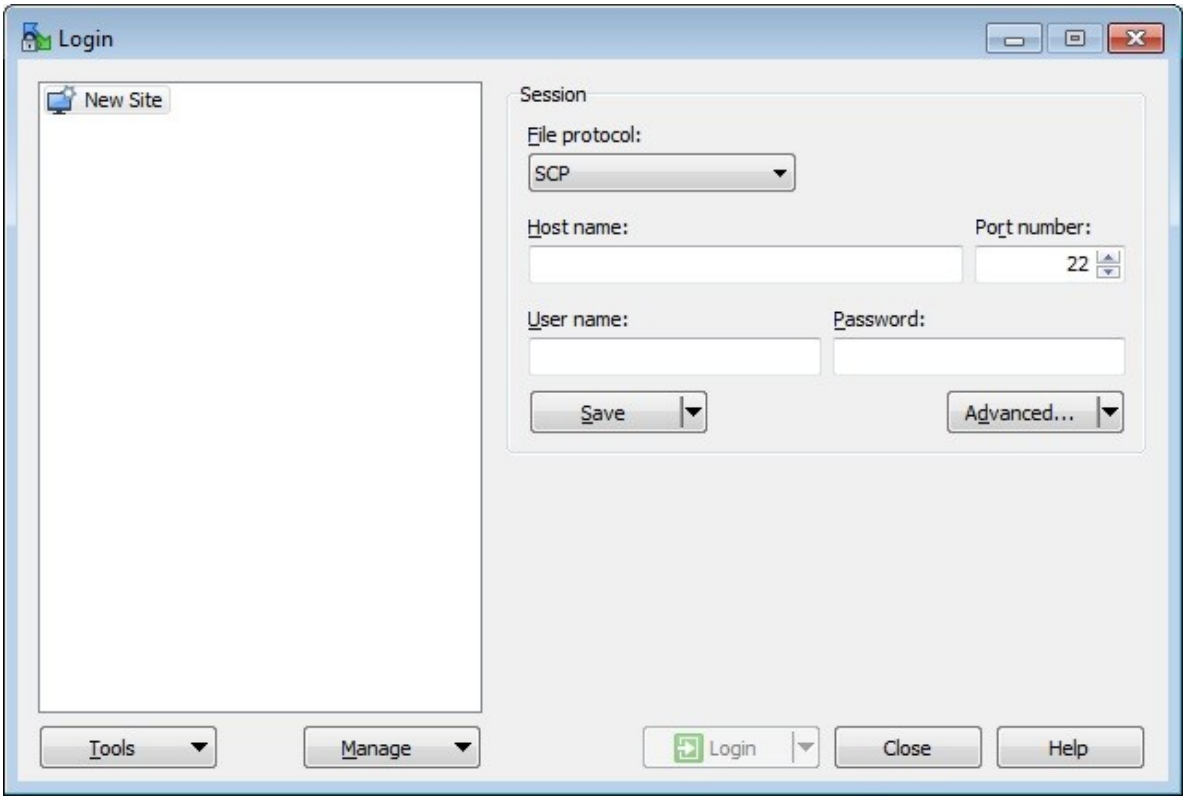
1.1) SCP en Windows

Sobre Windows se puede utilizar WinSCP. Descargar esta aplicación de https://winscp.net

Al ejecutar la aplicación nos abre la pantalla inicial:

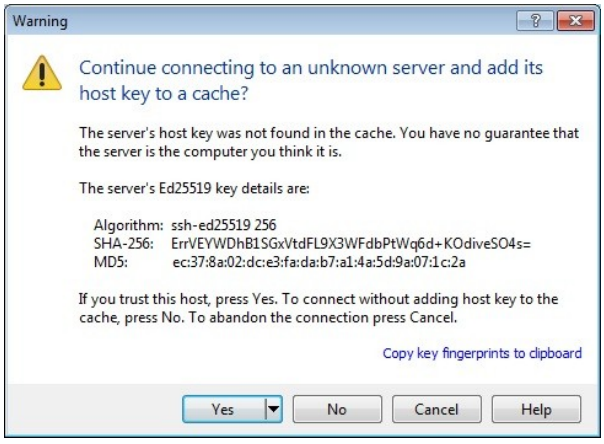


Debemos seleccionar SCP como protocolo de transferencias de archivos. Para esto desplegamos las opciones en **File protocol** y seleccionamos SCP. La pantalla inicial debería quedar entonces:



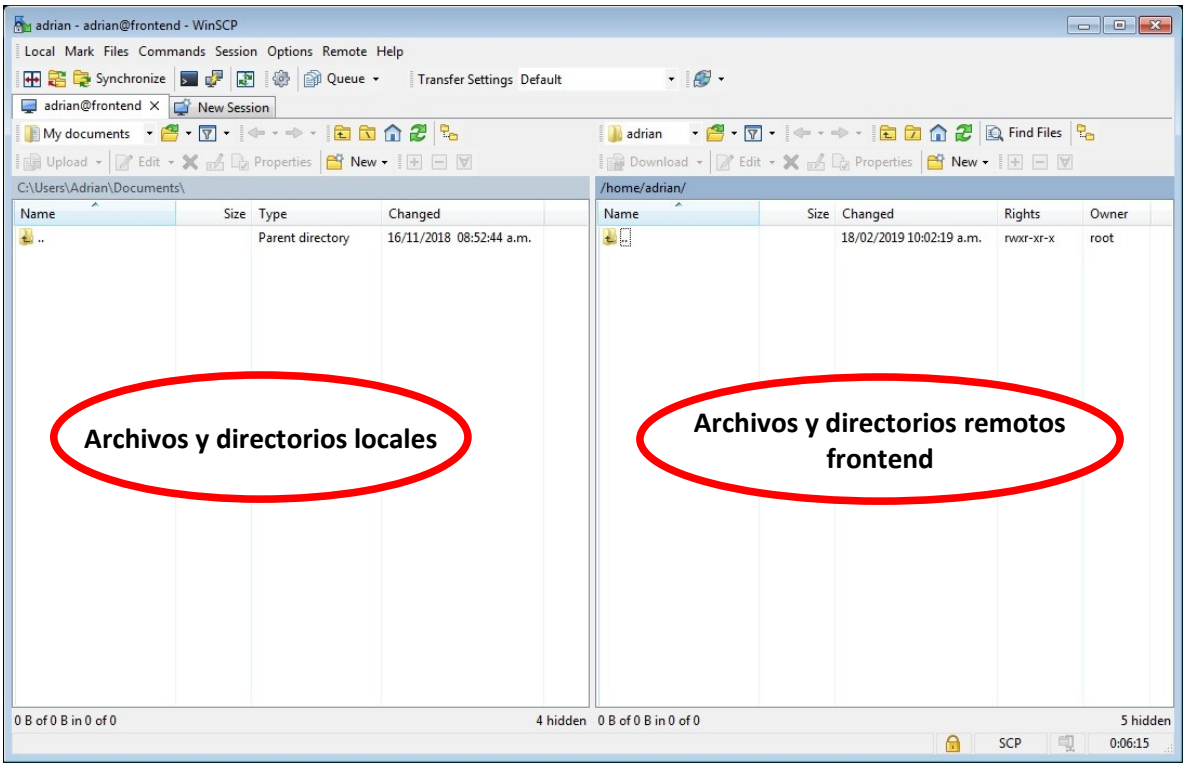
Luego, completar los campos de **Host name**, **User name** y **Password** de acceso al frontend con los datos dados oportunamente por la cátedra. Finalizar presionando el botón **Login**. Cabe destacar, que es posible guardar las conexiones (botón Save) para no realizar una y otra vez el procedimiento de conexión.

La primera vez que se accede es posible que la autenticación retorne una o dos ventanas de **Warning**:



Por cada ventana presionar el botón **Yes.**

Si la autenticación fue exitosa se abrirá un administrador de archivos:



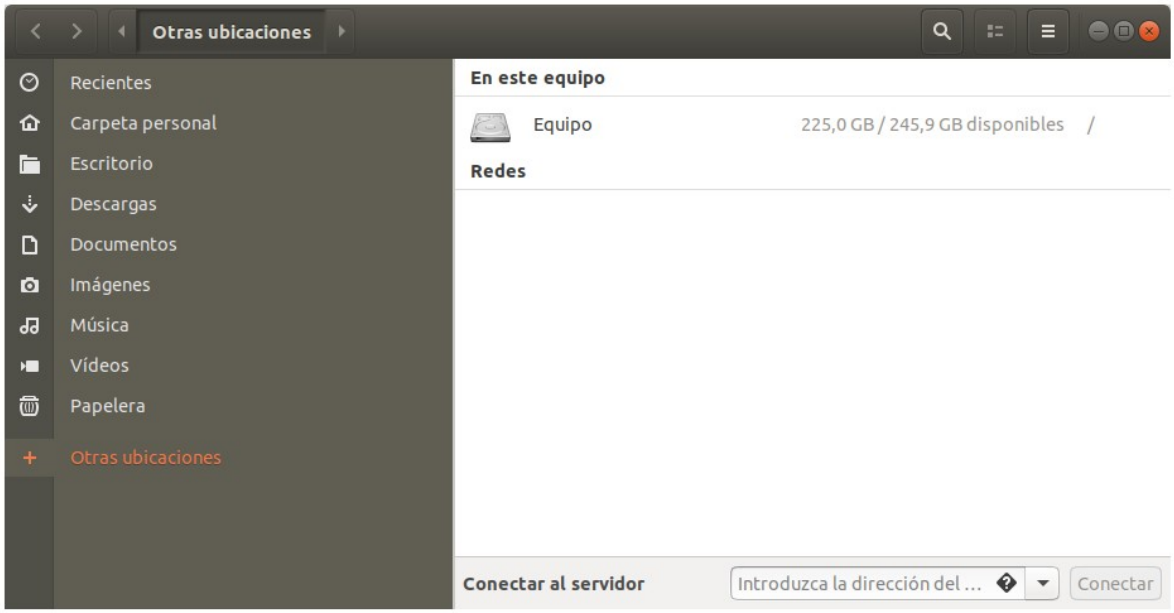
En el lado izquierdo muestra un panel donde se encuentran los archivos locales, en el lado izquierdo muestra otro panel donde se encuentran los archivos en el frontend.

Se pueden transferir archivos y directorios entre la máquina local y el frontend con sólo arrastrarlos con el mouse.

1.2) SCP en Linux

El uso de SCP varía de una distribución de Linux a otra. Para el ejemplo nos basaremos en la distribución de Linux Ubuntu.

Para transferir el código fuente de la aplicación hacia el cluster abrimos el navegador de archivos de Ubuntu (Nautilus). En el lateral izquierdo, vamos a **+ Otras ubicaciones**. Deberíamos ver algo como:



Luego completar el campo de la parte inferior, donce dice **Conectar al servidor** con la siguiente información:

`ssh://`**`HostNameFrontend/HOMEfrontendUSER`**

HostNameFrontend es el nombre de host del frontend.

`HOMEfrontendUSER` es el directorio del usuario con el que se ingresa. Este directorio será dado oportunamente por la cátedra.

Se abrirá una ventana en la que pedirá usuario y contraseña, se deberá ingresar el **usuario** y **contrañeña** del frontend.

Todos estos datos serán dados oportunamente por la cátedra.

Por último, presionar el botón conectar.

Una vez finalizado, el contenido del navegador de archivos será el contenido del directorio **HOMEfrontendUSER.**

Se pueden transferir archivos y directorios entre la máquina local y el frontend con sólo arrastrarlos desde una ventana del navegador de archivos local a la ventana del navegador de archivos en el frontend.

2) Acceder a la consola de texto y compilar

Una vez que se copiaron los archivos fuente al cluster es necesario acceder a una consola de texto en el frontend para compilar y posteriormente enviar a ejecutar.

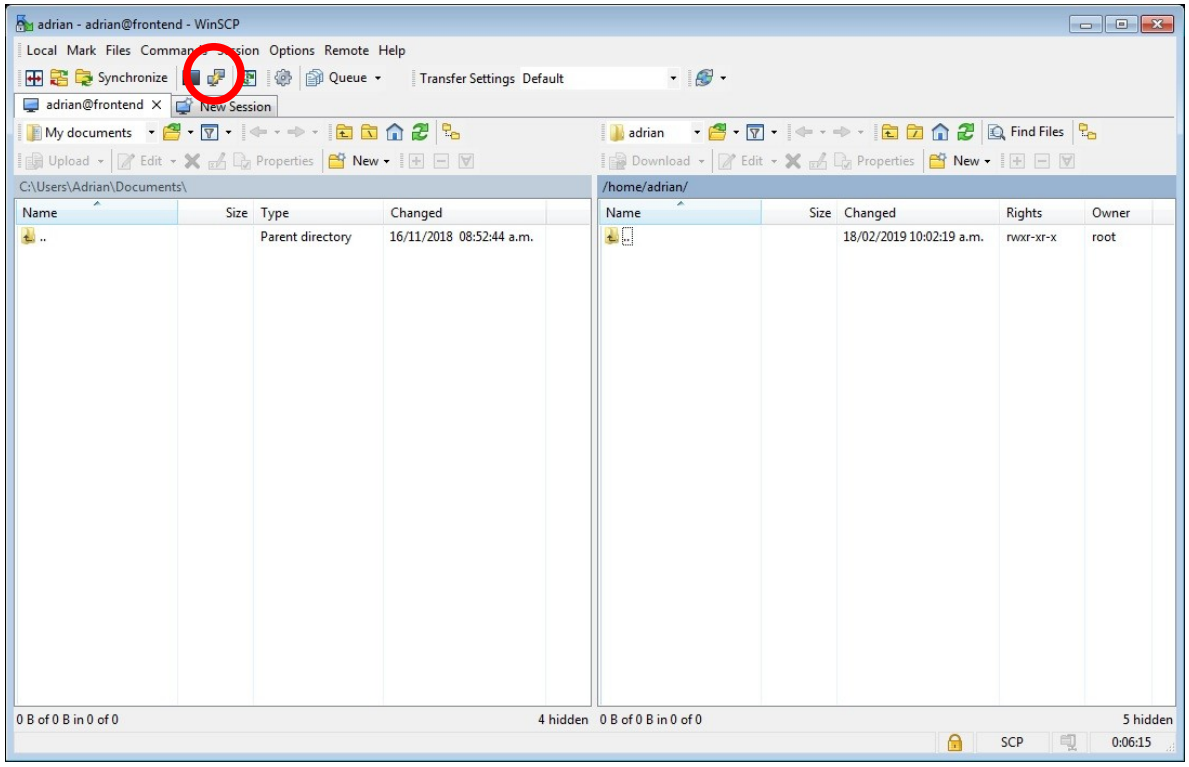
Para esto se accederá mediante el protocolo SSH. La forma de usar SSH varía al usar Windows o Linux.

2.1) SSH en Windows

Windows no trae un cliente SSH pero se puede descargar **Putty** del sitio https://www.chiark.greenend.org.uk

Se debe descargar el archivo **putty.exe** (NO el instalador).

Copiar el archivo putty.exe al directorio donde se instaló WinSCP. Luego, acceder al cluster a través de WinSCP como se indicó anteriormente. Se abrirá la ventana y se debe acceder al ícono (**Open Session in PuTTY**) que se indica en la siguiente imagen.



Se abrirá una consola de texto en la que pedirá contraseña. Se debe ingresar la contraseña de acceso al frontend.

Una vez autenticado se abrirá una consola de texto en el frontend.

2.2) SSH en Linux

En Linux, abrimos una consola de texto y escribimos:

```
ssh  UserNameFrontend@HostNameFrontend
```

Este comando nos pedirá la contraseña del usuario del frontend. Una vez finalizada la ejecución del comando nos encontramos en una sesión SSH en el frontend, es decir en la consola de texto en que ejecutamos el comando nos encontramos en una consola de texto en el frontend.

3) Ejecutar la aplicación a través del gestor de recursos

La ejecución de aplicaciones paralelas sobre el cluster NO debe hacerse directamente sino a través del gestor de recursos.

Para cada ejecución debe crearse un script para indicarle al gestor de recursos como ejecutar. Dependiendo de las características de la aplicación podemos considerar los siguientes casos:

- Aplicaciones secuenciales
- Aplicaciones paralelas de memoria compartida
- Aplicaciones paralelas de memoria distribuida

Cabe destacar, que la ejecución de la aplicación no es interactiva como en una ejecución tradicional. Una vez que finaliza la ejecución de la aplicación el gestor de recursos dejará dos archivos: uno con la salida de la aplicación y otro con la salida de errores (si es que los hay). Es importante indicarle al gestor de recursos el directorio y nombre de estos archivos.

3.1) Aplicaciones secuenciales

Crear un script con el siguiente contenido:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --exclusive
#SBATCH -o directorioSalida/output.txt
#SBATCH -e directorioSalida/errores.txt
./miAplicación
```

La línea **#SBATCH -N 1** le indica al gestor de recursos que debe reservar **un nodo (nodo1, nodo2, … nodoX)** del cluster y la línea **#SBATCH --exclusive** le indica que lo haga de forma exclusiva. Sin la línea de exclusividad, dos aplicaciones secuenciales de dos usuarios diferentes podrían ejecutar en el mismo nodo en cores diferentes.

Las líneas **#SBATCH -o directorioSalida/output.txt y #SBATCH -e directorioSalida/errores.txt** indican el directorio y archivo de salida y de errores, respectivamente. Es importante que el directorio de salida exista sino dará un error.

La última línea, **./miAplicación**, es el nombre de la aplicación a ejecutar. La linea comienza con ./ dado considerando que la aplicación se encuentra en el mismo directorio que el script. De no ser así, debe especificarse el directorio donde se encuentra la aplicación.

Si llamamos **miScript.sh** a nuestro script debemos darle permisos de ejecución. Para esto ejecutamos el siguiente comando:

```
chmod +x miScript.sh
```

Por último, ejecutamos nuestro script mediante el comando:

```
sbatch ./miScript.sh
```

Si la aplicación recibe parámetros es posible pasarlos a través del script:

```
sbatch ./miScript.sh par1 par2
```

dentro del script podemos referirnos a estos parámetros como:

```
./miAplicación $1 $2
```

Donde $1 será el primer parámetro pasado al script (par1), $2 será el segundo (par2) y así siguiendo.

Si hay recursos disponibles el administrador de recursos ejecutará la aplicación. Si no hay recursos disponibles la ejecución quedará pendiente y el requerimiento se encola como trabajo en una cola de trabajos pendientes.

Podemos ver el estado del trabajo mediante el comando:

squeue

que retorna una salida similar a:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
297	colaA	app1	usuario1	PD	0:00	1	(Resources)
298	colaB	app2	usuario2	PD	0:00	1	(Priority)
295	colaB	appMPI	usuario2	R	0:01	1	nodo1
296	colaA	appOMP	usuario1	R	0:01	1	nodo2

JOBID: identificador único del trabajo encolado.
PARTITION: el administrador de recursos puede manejar varias colas o particiones. Este campo indica en que cola fue encolado el trabajo.
USER: nombre del usuario propietario del trabajo.
ST: estado del trabajo. (R: running, PD: pending)
TIME: el tiempo que lleva de espera
NODES: el número de nodos solicitados
NODELIST (REASON): lista de nodos alocados

Podemos cancelar la ejecución de una aplicación o desencolar un trabajo encolado mediante el comando:

scancel JOBID

3.2) Aplicaciones paralelas de memoria compartida

Para el caso de aplicaciones **Pthreads** la creación de los hilos es explícita, La aplicación lanzará dos o más hilos que se ejecutarán sobre el nodo multicore.

Se debe crear un script:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --exclusive
#SBATCH -o directorioSalida/output.txt
#SBATCH -e directorioSalida/errores.txt
./miAplicación nroHilos
```

Uno de los parámetros de de la aplicación debe ser el número de hilos (**nroHilos**) con el que se quiera ejecutar. Luego, el programador debe crear ese número de hilos en el código.

Para el caso de **OpenMP** la creación de los hilos es implícita. Es el runtime system de OpenMP quien crea los hilos. Se debe crear un script con el siguiente contenido:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --exclusive
#SBATCH -o directorioSalida/output.txt
#SBATCH -e directorioSalida/errores.txt
export OMP_NUM_THREADS=8
./miAplicacionOMP
```

Sólo se debe agregar la línea **`export OMP_NUM_THREADS=8`** donde se indica el número de hilos con los que se quiere ejecutar.

3.3) Aplicaciones paralelas de memoria distribuida

Si queremos ejecutar una aplicación MPI tendremos varias alternativas:

- Ejecución sobre un único nodo
- Ejecución sobre varios nodos

3.3.1) Ejecución sobre un único nodo

Nos referimos a este tipo de ejecución cuando queremos ejecutar la aplicación MPI sobre un único nodo del cluster (**nodo1 o nodo2, … o nodoX**).

Crear un script con el siguiente contenido:

```
#!/bin/bash
#SBATCH -N 1
```

```
#SBATCH --exclusive
#SBATCH --tasks-per-node=8
#SBATCH -o directorioSalida/output.txt
#SBATCH -e directorioSalida/errores.txt
mpirun  miAplicacionMPI
```

En este caso, se agrega la linea #SBATCH –tasks-per-node=8 que indica que corra la aplicación en el único nodo con 8 procesos. Esto es equivalente a correr **`mpirun -np 8 miAplicacionMPI`**

3.3.2) Ejecución parcial sobre varios nodos

Nos referimos a este tipo de ejecución cuando queremos ejecutar la aplicación MPI sobre varios nodos del cluster (**nodo1 y nodo2, … y nodoX**). Por ejemplo, corremos 8 procesos sobre dos nodos, 4 en cada nodo.

Crear un script con el siguiente contenido:

```
#!/bin/bash
#SBATCH -N 2
#SBATCH --exclusive
#SBATCH --tasks-per-node=4
#SBATCH -o directorioSalida/output.txt
#SBATCH -e directorioSalida/errores.txt
mpirun miAplicacionMPI
```

En este caso, la línea **`#SBATCH -N 2`** indica que se deben reservar 2 nodos. La línea **`#SBATCH --tasks-per-node=4`** indica que deben ejecutarse 4 procesos por nodo. Esto es equivalente a correr **`mpirun -np 8 --map-by ppr:4:node miAplicacionMPI`**

3.4) Aplicaciones paralelas híbridas

Nos referimos a aplicaciones Híbridas cuando combinamos dos modelos de programación:

- Modelo de memoria distribuida MPI - Modelo de memoria compartida OpenMP.

- Modelo de memoria distribuida MPI - Modelo de memoria compartida Pthreads.

Vamos a ejecutar un proceso por cada nodo y cada proceso creará el número de hilos que requiera para ejecutar.

3.4.1) *Ejecución* MPI-OpenMP

La creación de los hilos es implícita. Por ejemplo, corremos 2 procesos sobre dos nodos, uno en cada nodo. Luego, cada proceso creará 4 hilos.

Crear un script con el siguiente contenido:

```
#!/bin/bash
#SBATCH -N 2
#SBATCH --exclusive
#SBATCH --tasks-per-node=1
#SBATCH -o directorioSalida/output.txt
#SBATCH -e directorioSalida/errores.txt
export OMP_NUM_THREADS=4
mpirun --bind-to none miAplicacionMPI
```

Es importante agregar el parámetro **`--bind-to none`**. Este parámetro resuelve un problema del Runtime System de OpenMPI evitando que todos los hilos se ejecuten sobre el mismo core.

3.4.2) *Ejecución* MPI-Pthreads

La creación de los hilos es explícita. Por ejemplo, corremos 2 procesos sobre dos nodos, uno en cada nodo. Cada proceso ejecutará **nroHilos** hilos.

Crear un script con el siguiente contenido:

```
#!/bin/bash
#SBATCH -N 2
#SBATCH --exclusive
#SBATCH --tasks-per-node=1
#SBATCH -o directorioSalida/output.txt
#SBATCH -e directorioSalida/errores.txt
mpirun --bind-to none miAplicacionMPI nroHilos
```

Uno de los parámetros de de la aplicación MPI debe ser el número de hilos (**nroHilos**) con el que se quiera ejecutar. Luego, el programador debe crear ese número de hilos en el código.

4) Recuperar los resultados

Los resultados de cada ejecución quedarán en los archivos de salida que le indiquemos al gestor de recursos. Para recuperarlos del cluster sólo tenemos que utilizar el protocolo SCP de manera inversa.