Práctica

0

Multiprocesadores

Utilización de librerías de paso de mensajes. MPI.

En esta práctica se instalará la implementación de MPI adecuada.

Ejercicio 1. Instalación portátil

Actualmente existen dos implementaciones vigentes de la librería MPI: MPICH la clásica y open MPI la moderna. En el laboratorio se tiene instalada esta última.

Si no lo has hecho, procede a instalarla en tu máquina personal.

En el caso de sistemas no Ubuntu tienes en http://www.open-mpi.org/software/ompi/v1.4/u una versión para sistemas debían o red hat (usa o bien tgz o bien rpm). Una macro autogen lo instala pero son necesarios autoconf, automake y libtool. También se puede hacer:

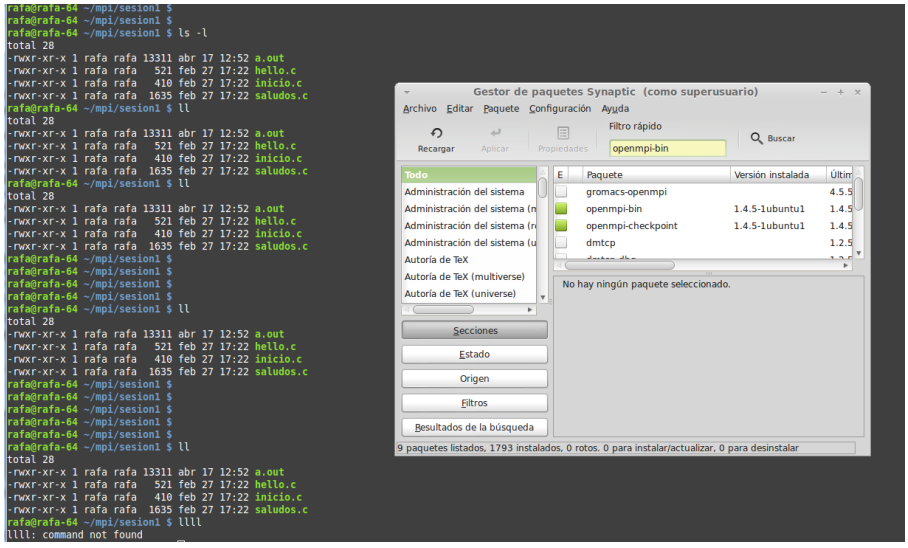
```
$ ./configure –prefix=/usr/local (en /usr/local)

# make all install
```

En el caso de Ubuntu intenta ejecutar mpicc, te dirá el paquete que te tienes que bajar:

```
sudo apt-get install openmpi-dev
```

O desde Synaptic (o el instalador de software):



Una vez instalado en tu portátil, bájate el programa de ejemplo y ejecútalo para ver si tienes el compilador de mpi y el lanzador: `mpicc` y `mpirun`. Ejecútalo con el mismo número de procesos que *cores* tengas.

A continuación, cambia el programa para usar algo equivalente a “hello world” con la primitiva:

```
MPI_Get_processor_name  // debes saber ya donde encontrar su prototipo
```

Ejercicio 2. Probar plataforma ssh (obsoleta no disponible 2021)

Una vez comprobado que tu sistema funciona, prueba a usar el laboratorio con el mismo programa, para ello deberás ejecutar los siguientes pasos:

1. Entra en tu usuario (no el genérico).
2. Genera la clave pública con: `ssh-keygen –t rsa`.

   Esto creará el directorio “`.ssh`” en el que habrá dos ficheros nuevos “`id_rsa`” y “`id_rsa.pub`”. Es aconsejable dejar vacía la *password* cuando la pida.
3. Copia la clave pública (`id_rsa.pub`) al fichero de autorizaciones: “`authorized_keys`”
4. Desde este punto ya se podrá acceder desde nuestro puesto a otras máquinas utilizando ssh. La primera vez pedirá confirmación (yes), las siguientes no ya que habrá quedado registrado en el fichero “`known_hosts`”.
5. Para saber los nombres (ip) de las máquinas disponibles (es necesario que estén encendidas y con Linux) existe una macro llamada “`machine_list`” que nos facilitará la vida dándonos sus ip en forma de lista. Por lo tanto, podemos hacer una macro (o una línea de comando compleja) para entrar en todas las máquinas del laboratorio para siempre (ojo con los apóstrofes invertidos):

   ```
   for IP in `machine_list` ; do ssh $IP echo; done
   ```
6. Si nuestra interface de red no es la de por defecto, se puede poner la nuestra como argumento a `machine_list`. (ver `ifconfig` o `ip`). Actualmente no hace falta.
7. Los anteriores puntos sólo hay que hacerlos una vez, salvo catástrofe. Lo que deberemos hacer cada vez que abramos sesión es:
   a. Arrancar el agente de ssh: `eval `ssh-agent``
   b. Darle la clave: `ssh-add $HOME/.ssh/id_rsa`
   c. Si ya estás en disposición de usar las 12 máquinas (96 cores físicos) del laboratorio III, puede realizar un fichero de host para poder ejecutar el programa de pruebas con esos procesadores/procesos. Para ello debes usar lo que te devuelva la macro `machine_list` en un fichero y modificarlo para que tenga un aspecto parecido a esto:

      ```
      IP1   slots=8
      IP2   slots=8
      IP3   slots=8
      …
      ```

En este caso también te puedes hacer una macro para realizarlo automáticamente. Hay un modelo en Moodle. Al menos, entiéndela y modifícala a tu gusto.

Por último, deberás compilar con `mpicc` y ejecutar con el fichero host:

```
mpirun -hostfile mifichero -np numero_de_cores nombre_ejecutable
```

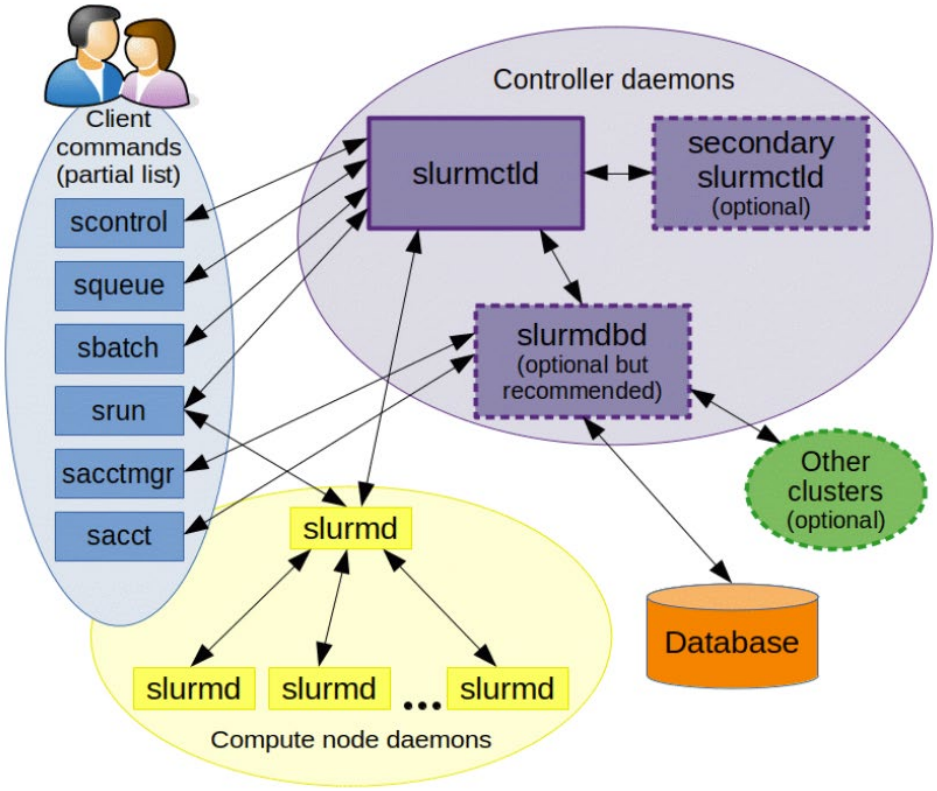
Ejercicio 3. Plataforma cluster de PC con *slurm*[1]

Este curso 21/22 se ha creado un sistema RTE con *slurm* en que los PC de los laboratorios pueden ser usados como una máquina multiprocesadora de varios nodos. De esta manera habrá PC que sirvan de clientes para entrar en el multiprocesador virtual y PC que sirvan de nodos.

El sistema gestor de colas está basado en *SLURM* que es el más usado en el TOP 500 (60%). Es un software abierto con una gran comunidad de usuarios detrás.

Para acceder al clúster debes acceder con "`ssh selkie`". Los nodos de cómputo se arrancan con el tercer método de entrada (Ubuntu, Windows, **Selkie**) y ya te les encontrarás encendidos en caso de uso.

Slurm permite ver a un conjunto de PC unidos por red como un clúster Beowulf. Cada nodo de cómputo (figura) tiene un demonio (*slurm*d) y existe un nodo de control central con su propio demonio de control (*slurm*ctld). Los clientes (usuarios) pueden ejecutar sus trabajos desde cualquier nodo y tienen acceso al sistema a través de una serie de comandos: `sacct, sacctmgr, salloc, sattach, sbatch, sbcast, scancel, scontrol, scrontab, sdiag, sh5util, sinfo, sprio, squeue, sreport, srun, sshare, sstat, strigger y sview` (no todos tienen que estar instalados). La arquitectura de slurm genérica es la siguiente:



El cliente cuando lance sus trabajos puede escoger reservar en exclusiva (o no) una serie de recursos durante un periodo de tiempo. Además, puede ver y controlar como sus trabajos evolucionan. *Slurm* se encargará de gestionar todos estos trabajos y hacer todas las labores de contención necesarias.

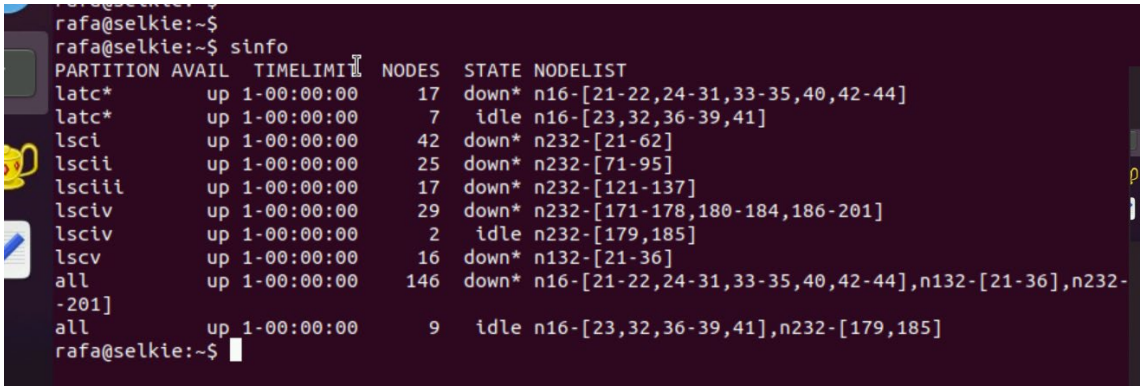
Las entidades que maneja *slurm* son nodos, particiones (grupos de nodos) y trabajos (o partes de un trabajo -steps-). Con los comandos vistos se pueden asignar los trabajos a los recursos y arrancar su ejecución de forma o exclusiva o no exclusiva.

[1] https://*slurm*.schedmd.com/

Puedes encontrar guías de uso de *slurm* en muchos sitios (https://slurm.schedmd.com/pdfs/summary.pdf), pero los comandos más utilizados son:

- sinfo: obtiene información de estado del cluster.
- squeue: obtiene el estado de las colas.
- srun: ejecuta un trabajo.
- sbatch: ejecuta un script.
- scancel: cancela un trabajo lanzado.
- sacct: obtiene estadísticas de uso del supercomputador.

Veamos cómo sería una sesión típica de trabajo. Una vez que estamos en *selkie* que es nuestro clúster, podemos ver los nodos que tenemos disponibles:



Como veis las particiones son los propios laboratorios de la facultad y podemos lanzar trabajos de MPI a las particiones que estén en la misma subred (n1 o n2). Los nombres de los nodos tienen el formato de nxx-yy. El comando nos dirá cuantas particiones hay, que nodos están activos o caídos (estado), y la lista de los mismos.

El sistema de ficheros será el mismo que tenéis en el laboratorio usual y podrás usar el comando de compilación de MPI de la forma usual. Para ejecutar un programa deberás usar los comandos de slurm o bien de forma interactiva o bien (lo recomendable) a través de batch.

En modo interactivo se usa el comando *srun* (en nuestro caso **prun**) al que se le puede dar el tiempo total de ejecución del trabajo, la cantidad de cores usada o la cantidad de memoria entre otras muchas cosas. Este modo también es llamado de debug, pero en nuestro caso en más sencillo usar el propio Linux (máquina) de donde venimos, por lo que **no** se debería usar.

```
srun --job-name=mitrabajo --time=00:20:00 --partition=latc --ntasks=1 --mem=512 --pty bash -l
```

Pero como hemos dicho, es recomendable usar el modo *batch*, para lo cual tenemos que crear una macro -script- de ejecución del trabajo y lanzarla con el comando **sbatch**. Un ejemplo de macro sería:

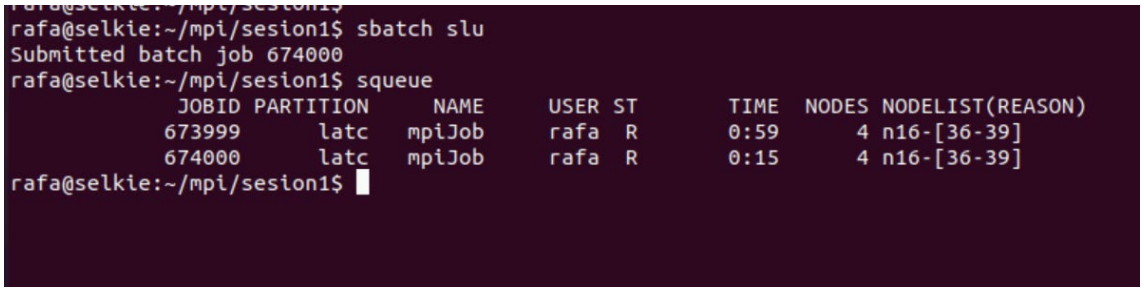
```
#!/bin/bash
#SBATCH --partition=latc        #especifica la partición grupo de nodos

#SBATCH -o mpi.%j.out           #Nombre del archivo de salida
#SBATCH -J mpiJob               #Nombre del trabajo
#SBATCH --nodes=4               #Numero de nodos para correr el trabajo
#SBATCH --ntasks=4              #Numero de procesos
#SBATCH --tasks-per-node=1 #Numero de tareas por nodo
```

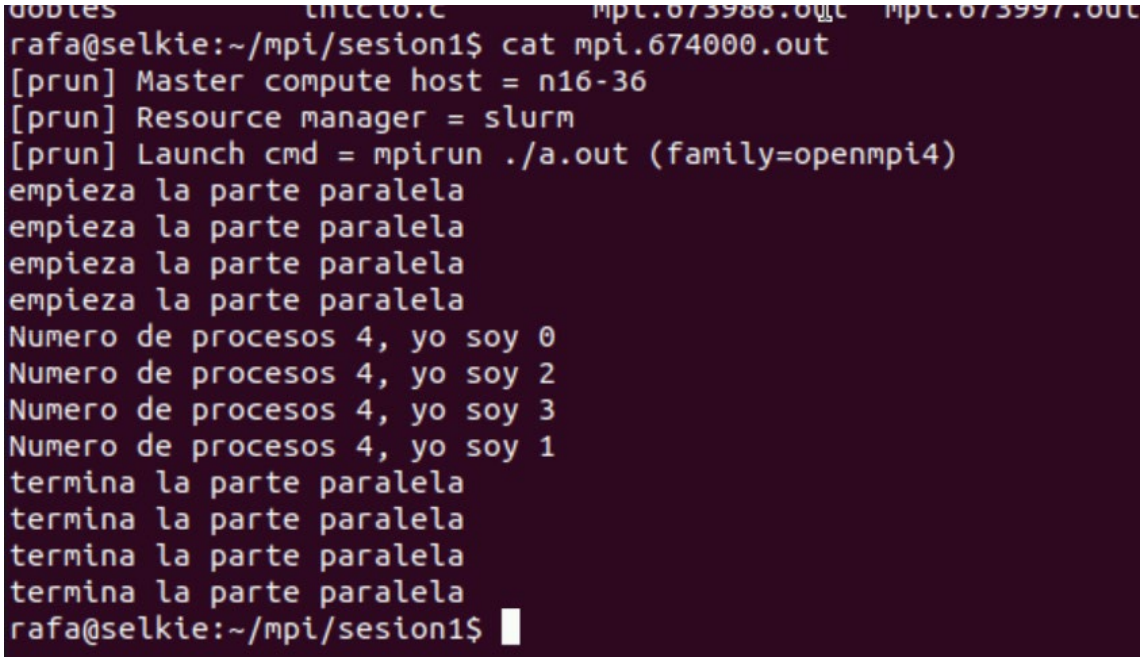
```
#Prepara el ambiente de trabajo(no es necesario)
export I_MPI_PMI_LIBRARY=/usr/local/slurm/lib/libpmi.so
ulimit -l unlimited

#Ejecuta el programa paralelo (en este cluster no se usa srun)
prun ./a.out
```

A continuación, bastaría con ejecutar la macro con el comando **sbatch** que nos responderá con el nombre del trabajo y si es duradero podremos ver el mismo con el comando **squeue**, si no acabará también se puede cancelar con **scancel**:



Ya que se trata de un trabajo en *batch*, los resultados se mostrarán en el archivo indicado.



Con el comando *module list* podréis ver los módulos cargados, entre ellos están el **prun** y el **openmpi** y si no hubiera cargado algo, como se verá en el siguiente apartado, se podría cargar con *module load*.

Ejercicio 4. Altamira (equivalente a la anterior, solo si no funciona 3)

El último entorno donde podéis ejecutar programas MPI es en el supercomputador Altamira (https://confluence.ifca.es/display/IC/Altamira+Users+Guide) de manera simplificada, ya que realmente para programa grandes se debería usar el sistemas de colas *SLURM* (https://*slurm*.schedmd.com/). Los pasos para la ejecución son:

1. Deberéis entrar en el sistema con `ssh usuario@altamira1.ifca.es` y poner vuestro password. Lo podéis cambiar.

2. Las siguientes veces se puede entrar en otro nodo menos descargado `ssh usuario@altamira2.ifca.es`

3. Una vez registrados, podéis copiar ficheros a Altamira con el comando `scp` de la forma habitual, señalando el fichero origen y copiándolo en la máquina destino: `scp fichero.c usuario@altamira2.ifca.es:fichero.c`

4. Entrad el Altamira y cargad el módulo de openmpi: `module load OPENMPI` dará algún aviso que se puede ignorar, comprobad que tenéis el módulo cargado: `module list`.

5. Después podéis proceder como siempre con los comandos `mpicc` y `mpirun`.