Documentación Prácticas CESGA

Integración del virtualizador *Singularity* con *openMPI* e *Infiniband* en el Supercomputador Finisterrae 2 (FT2)

Índice

Copiado de ficheros a FT2

Para mover ficheros desde nuestra máquina local al FT2 usaremos *scp*. Tener en cuenta que después de los dos puntos se indica el directorio donde queremos copiar las cosas. Lo normal es copiarlo en *LUSTRE*.

```
scp openmpi.img tec_sis4@ft2.cesga.es:/mnt/lustre/scratch/home/cesga/tec_
sis4
```

> **Nota:**
> `$LUSTRE=/mnt/lustre/scratch/home/cesga/tec_sis4`

Conexión mediante SSH con FT2

La conexión con el Supercomputador puede ser realizada desde nuestra máquina local de la siguiente manera:

```
ssh tec_sis4@ft2.cesga.es
```

Utilización del FT2

Movimiento al directorio adecuado

Una vez inicilizada la sesión, debemos tener en cuenta una serie de aspectos para trabajar con el supercomputador.
Para poder disfrutar del máximo espacio, debemos movernos al directorio *LUSTRE*, donde existen varios TB disponibles.

```
cd $LUSTRE
```

Carga de módulos

Si intentamos emplear el FT2 directamente, por ejemplo, ejecutando el siguiente contenedor:

```
./openmpi.img
```

Nos saltará un error, puesto que antes de poder emplearlo **debemos cargar los módulos que**

nos sean necesarios en nuestra cuenta del FT2. Así, para poder cargar contenedores con Singularity será necesaria la carga de su módulo correspondiente:

```
module load singularity
```

Tener en cuenta que para la tarea a desempeñar, **no basta con** la utilización de ***Singularity***, sino que es muy **importante** su integración con ***openMPI***. Así, será necesario cargar también su módulo, aunque en principio no se nos indique ningún error (los programas correrían sin el uso de Infiniband y el paso de mensajes entre procesadores).

Coexistencia de diferentes módulos para diferentes versiones de la misma utilidad dentro del FT2

En el FT2 coexisten diferentes módulos de *openMPI*, y de otras utilidades, como puede ser el compilador *gcc*. Es por ello, que **debemos consultar las diferentes versiones** disponibles en el mismo y **cargar aquella que nos interese** en determinado momento. Para dicha función, contamos con el comando ***spider***, que nos indicará todas las versiones existentes de los diferentes módulos a buscar. Por ejemplo, si queremos cargar un módulo de *openMPI*, primero debemos hacer la consulta *spider*.

```
module spider openmpi
```

Una vez desplegada la lista de versiones y decidida cuál queremos emplear, podemos cargar el módulo correspondiente. Por ejemplo:

```
module load openmpi/1.10.2
```

Es posible (y muy probable) que **al cargar algún módulo** nos salte algún tipo de **error debido a dependencias** con otras librerías o módulos. En este caso, se nos indicará las librerías de las cuales depende, y **debemos cargarlas también**. En principio, no hará falta el uso de *spider*, pues ya se nos indicará las distintas versiones del módulo disponible que deberían funcionar para subsanar la dependencia.
En cuanto tengamos decidida la versión del módulo del que depende que queremos cargar, podemos lanzar ambos módulos al mismo tiempo para que todo funcione correctamente. Por ejemplo:

```
module load gcc/5.3.0 openmpi/1.10.2
```

Hecho esto, ya deberíamos tener nuestro **entorno completamente listo** para poder trabajar con ***Singularity*** y su **integración con *openMPI*, haciendo uso de *Infiniband***.

Consulta de los módulos que tenemos cargados actualmente:

```
module list
```

Lanzamiento de aplicaciones en el entorno

Una vez tengamos nuestro entorno preparado, podemos lanzar las diferentes aplicaciones y pruebas que deseemos hacer. **Tener en cuenta** que se lanzarán según los **recursos** que tengamos **disponibles**, según solicitemos. Para ello, debemos seguir el siguiente modelo:

```
mpirun -np 2 singularity exec openmpi.img ring
```

> El comando anterior nos lanzará con *MPI*, haciendo uso de 2 procesadores, y dentro del contenedor *openmpi.img*, el programa *ring*.
>
> - El **código fuente** del programa **ring**, implementado en *C*, se puede consultar en: https://raw.githubusercontent.com/open-mpi/ompi/master/examples/ring_c.c
>   Debemos descargarlo y compilarlo cuantas veces sea preciso.
> - El programa ring es un programa de ejemplo incluido en el proyecto OpenMPI, que se puede consultar en su totalidad en el repositorio: https://github.com/open-mpi/ompi, en el apartado *examples*.

Reserva de recursos en el FT2

Cuando hacemos uso de la aplicación y las pruebas, debemos emplear diferentes recursos. Para ello, **debemos indicarle al FT2 cuanto deseamos consumir**, para que nos los ceda una vez estén disponibles. Se puede hacer de la siguiente forma:

```
compute -c 4
```

> Esto nos ortorgará 4 procesadores en cuanto sea posible.

Es posible que al hacer este cambio de contexto, los módulos que teníamos cargados antes, dejen de estarlo (en principio, no debería, pero más vale prevenir). Es por esto que debemos desenlazar los módulos cargados y volver a cargarlos.

```
module list
module purge
module load gcc/5.3.0 openmpi/1.10.2 singularity/2.2.1
```

> **Listamos para no olvidar -> Desenlazamos -> Recargamos**

Ahora que tenemos los recursos reservados, podríamos pensar que ya es posible lanzar el

programa pertinente (p.e. *ring*) con el nº de procesadores deseados y que **todo funcionara** correctamente, pero esto **no** es así, puesto que surjen problemas con los directorios dentro y fuera del contenedor (fuera existen directorios con los que se intenta trabajar que no están dentro del contenedor).
Por lo tanto lo siguiente **NO funcionará**:

```
mpirun -np 4 singularity exec openmpi.img ring
mpirun -np 2 singularity exec openmpi.img ring
mpirun -np 1 singularity exec openmpi.img ring
```

En caso de intentar la **llamada directamente al contenedor**, sin tener lugar la ejecución con *mpi (mpirun)*, **SÍ nos funcionará** con 1 procesador, pero esto no tiene interés:

```
singularity exec openmpi.img ring
```

Para subsanar este **error, ya controlado,** debemos pasarle el directorio para que lo tome de forma adecuada:

```
mpirun -np 1 singularity exec -B /scratch:/scratch openmpi.img ring
```

Para simplificar las llamadas y que sean más directas, existe ya un script, implementado en bash que nos simplificará el trabajo. Este es:

> **run_singularity.sh**
>
> ```
> /opt/cesga/singularity/2.2.1/utils/run_singularity.sh
> ```

Si estamos dentro del *compute* (fijarse en los números de la sesión, donde debería estar el nombre de usuario a la izda), hacemos un *exit* y desde la anterior, lanzamos directamente con *run_singularity*.
Si están todos los recursos listos y los módulos, ahora sí podremos lanzar todo correctamente con:

```
mpirun -np 2 run_singularity.sh openmpi.img ring
```

En caso de no indicar como argumentos el número de procesadores, tomará el máximo que le sea posible, que en nuestro caso de prueba concreto, serán 24:

```
mpirun run_singularity.sh openmpi.img ring
```

Cosas a tener en cuenta si estamos dentro de los contenedores

Para no equivocarnos entre el contenedor el propio FT2, podemos comprobar la información esencial del SO.

```
cat /etc/lsb-release
```

Para **salir** de un contenedor, lo hacemos con ***exit* o *Ctrl+D***.

Pruebas a realizar dentro del FT2

Una vez comprendido el funcionamiento, debemos comenzar a realizar las diferentes **pruebas**, hasta conseguir una **correlación entre los fallos y las diferentes versiones de *Singularity* y *openMPI***. Para ello, haremos uso de diferentes contenedores, con diferentes *SO* base, que tendrán diferentes versiones de *openMPI*.
Debemos tener en cuenta que desde las indicaciones de *Singularity*, se nos asegura que el uso de una versión inferior, dentro de un contenedor, de *openMPI* (u otro controlador de *MPI*) debe ser completamente compatible con el exterior del contenedor (en este caso, el FT2) si este posee una versión igual o superior. Solo se ha observado este correcto comportamiento con exactamente la misma versión, pero no con diferentes versiones (inferior -> contenedor, superior -> FT2), por lo que se debe investigar la causa de este posible problema.
Así, iremos variando entre los módulos disponibles en el FT2 y verificando:

```
module load gcc/6.1.0
```

> Al hacer esto, ya se desenlazará el módulo anteriormente cargado y se meterá el indicado.

Ejemplo de una prueba completa

```
[tec_sis4@fs6803 tec_sis4]$ module spider openmpi/1.10.2

----------------------------------------------------------------------------
----------------------------------------------------------------
  openmpi: openmpi/1.10.2
----------------------------------------------------------------------------
----------------------------------------------------------------

    This module can only be loaded through the following modules:

      gcc/5.3.0
```

```
      gcc/6.1.0
      intel/2016


[tec_sis4@fs6803 tec_sis4]$ module load g
g09                     gatk/3.6               gcc/6.3.0              git
gmt/4.5.14              grib_api
g09/d1-mkl              gatk/3.7               gdal                   git/
2.7.1              gmt/5.2.1              grib_api/1.14.5
g09/e1                  gblocks                gdal/1.11.4            gl2p
s                  gnuplot                gsl
g2clib                  gblocks/0.91b          gemtools               gl2p
s/1.3.9            gnuplot/5.0.3          gsl/1.16
g2clib/1.5.0-patch      gcc                    gemtools/1.7.1-i3      gmp
gnuplot/5.0.4
gadget                  gcc/4.4.7              genometools            gmp/
6.1.0              gperftools
gadget/2.2.00-BETA      gcc/4.8.2              genometools/1.3.7      gmsh
gperftools/2.5
gatk                    gcc/4.9.1              geos                   gmsh
/2.16.0            graphicsmagick
gatk/3.5                gcc/6.1.0              geos/3.5.0             gmt
graphicsmagick/1.3.24
[tec_sis4@fs6803 tec_sis4]$ module load gcc/6.1.0
gcc/5.3.0 unloaded
openmpi/1.10.2 unloaded
gcc/6.1.0 loaded
openmpi/1.10.2 loaded

Due to MODULEPATH changes the following have been reloaded:
  1) openmpi/1.10.2

The following have been reloaded with a version change:
  1) gcc/5.3.0 => gcc/6.1.0

[tec_sis4@fs6803 tec_sis4]$ mpirun run_singularity.sh openmpi.img ring
Process 0 sending 10 to 1, tag 201 (24 processes in ring)
Process 0 sent to 1
Process 10 exiting
Process 11 exiting
Process 12 exiting
Process 13 exiting
Process 14 exiting
Process 15 exiting
Process 16 exiting
Process 17 exiting
Process 18 exiting
Process 19 exiting
Process 20 exiting
Process 21 exiting
Process 22 exiting
Process 23 exiting
```

```
Process 0 decremented value: 9
Process 0 decremented value: 8
Process 0 decremented value: 7
Process 0 decremented value: 6
Process 0 decremented value: 5
Process 0 decremented value: 4
Process 0 decremented value: 3
Process 0 decremented value: 2
Process 0 decremented value: 1
Process 0 decremented value: 0
Process 0 exiting
Process 1 exiting
Process 2 exiting
Process 3 exiting
Process 4 exiting
Process 5 exiting
Process 6 exiting
Process 7 exiting
Process 8 exiting
Process 9 exiting
[tec_sis4@fs6803 tec_sis4]$ module spider openmpi/1.10.2

----------------------------------------------------------------------------
----------------------------------------------------------------
  openmpi: openmpi/1.10.2
----------------------------------------------------------------------------
----------------------------------------------------------------

    This module can only be loaded through the following modules:

      gcc/5.3.0
      gcc/6.1.0
      intel/2016


[tec_sis4@fs6803 tec_sis4]$ module load intel/2016
intel/2016 loaded

Lmod is automatically replacing "gcc/6.1.0" with "intel/2016"

gcc/6.1.0 unloaded
intel/2016 unloaded
openmpi/1.10.2 unloaded
intel/2016 loaded
openmpi/1.10.2 loaded

Due to MODULEPATH changes the following have been reloaded:
  1) openmpi/1.10.2

[tec_sis4@fs6803 tec_sis4]$ mpirun run_singularity.sh openmpi.img ring
Process 21 exiting
Process 22 exiting
```

```
Process 23 exiting
Process 0 sending 10 to 1, tag 201 (24 processes in ring)
Process 0 sent to 1
Process 0 decremented value: 9
Process 0 decremented value: 8
Process 0 decremented value: 7
Process 0 decremented value: 6
Process 0 decremented value: 5
Process 0 decremented value: 4
Process 0 decremented value: 3
Process 0 decremented value: 2
Process 0 decremented value: 1
Process 0 decremented value: 0
Process 0 exiting
Process 1 exiting
Process 2 exiting
Process 3 exiting
Process 4 exiting
Process 5 exiting
Process 6 exiting
Process 7 exiting
Process 8 exiting
Process 9 exiting
Process 10 exiting
Process 11 exiting
Process 12 exiting
Process 13 exiting
Process 14 exiting
Process 15 exiting
Process 16 exiting
Process 17 exiting
Process 18 exiting
Process 19 exiting
Process 20 exiting
```

Resultados de las pruebas realizadas

V. *OpenMPI* Ft2 → **——————————** **V. *OpenMPI* contenedor ↓**	**1.10.2**	**2.0.0**	**2.0.1**	**2.0.2**	**2.1.1**
1.10.2	-	-	-	-	-
2.0.0	-	-	-	-	-
2.0.1	-	-	-	-	-

V. *OpenMPI* Ft2 → —————————— V. *OpenMPI* contenedor ↓	1.10.2	2.0.0	2.0.1	2.0.2	2.1.1
2.0.2	-	-	-	-	-
2.1.1	-	-	-	-	-