Guía para el uso de la supercomputadora Miztli

Coordinación de Supercómputo
Dirección General de Tecnologías de Información y Comunicación
UNAM

Índice

La presente guía contiene información básica para utilizar la supercomputadora Miztli. Si requiere mayor información, envíe un mensaje de correo electrónico a ayuda@super.unam.mx

1. Descripción general del equipo

La supercomputadora Miztli es una computadora que cuenta con diferentes tipos de recursos de procesamiento:

Nodos regulares
Constituyen la cantidad más grande de cores de la supercomputadora, por lo que están dedicados a efectuar la mayor parte de los cálculos que en ella se realizan. En total son 430 nodos con procesadores Intel Xeon E5, que suman 8040 cores, contenidos en tres tipos de nodos regulares, pero agrupados en cinco grupos distintos:

- **g1**. Contienen dos procesadores E5-2670v1, un total de 16 cores y 64 Gbytes de RAM.
- **g2_a** y **g2_b**. Contienen dos procesadores E5-2660v3, un total de 20 cores y 128 GBytes de RAM.
- **g3_a** y **g3_b**. Contienen dos procesadores E5-2683v4, un total de 32 cores y 256 GBytes de RAM.

Nodos 256g
Son 10 nodos con procesadores Intel E5-2670v1 y 256 GBytes de RAM, para un total de 160 cores destinados a los cálculos que requieren una cantidad excepcional de memoria RAM.

Nodos GPU
Son 8 nodos con procesadores Intel E5-2670v1 y 2 tarjetas NVIDIA M2090. En total son 128 cores y 16 tarjetas M2090 dedicados a los cálculos que requieren el uso de GPUs (programas hechos con CUDA Toolkit).

Todos los tipos de nodos deben utilizarse a través del sistema de colas de Miztli. En la Sección 6 se muestran los comandos básicos y algunos ejemplos.

2. Conexión a Miztli

Para acceder a la supercomputadora Miztli, se debe acceder en primera instancia a la VPN de la Coordinación de Supercómputo, posteriormente se realiza una conexión a la supercomputadora mediante un cliente de SSH. En caso de que su máquina se encuentre dentro de Red UNAM, puede realizar directamente la conexión de SSH. Si requiere información adicional acerca del acceso a la supercomputadora, por favor comuníquese a la Coordinación de Supercómputo.

3. Espacios de almacenamiento

Cada usuario tiene tres espacios de almacenamiento distintos en Miztli:

- **TEMPORAL (TMPU).** Es el espacio de mayor capacidad y el único disponible para la ejecución de sus jobs. No está restringido, pero es depurado de forma automática y periódica.
- **PERMANENTE (BAUL).** Es un espacio restringido en el que pueden almacenarse archivos de forma permanente.
- **HOME**. Es un espacio dedicado a los archivos de configuración de cada cuenta.

Cada uno de estos espacios está implementado en diferentes sistemas de almacenamiento (`/tmpu`, `/baul1` y `/baul2`, `/home`, respectivamente). Puede consultar información acerca de ellos mediante el comando `df`:

```
% df -lh
Filesystem             Size  Used Avail Use% Mounted on
...
nfs1_ib:/home          461G   37G  400G   9% /home
nfs1_ib:/baul1          12T  2.2T  9.0T  20% /baul1
nfs1_ib:/baul2          12T  890G   11T   8% /baul2
172.20.2.101@o2ib:172.20.2.100@o2ib:/hpcfs
                       219T  142T   75T  66% /tmpu
...
```

Otras características de los espacios de almacenamiento son las siguientes:

- **TMPU**
  - `/tmpu` es el sistema de almacenamiento principal del equipo de supercómputo Miztli.
  - Está implementado en un sistema de archivos tipo Lustre, paralelo, de alto rendimiento, exportado a través de una red Infiniband.
  - En TMPU debe almacenarse la información que se genera en el tiempo de ejecución de los programas, tales como archivos temporales o de scratch.
  - No hay límites para el uso de TMPU, pero los archivos no pueden almacenarse permantemente ahí.
  - Si desea conservar algunos archivos almacenados en TMPU, debe moverlos a BAUL o hacer un respaldo en su máquina local.

  Para acceder a su directorio en /tmpu, utilice la variable de ambiente TMPU:

  ```
  cd $TMPU
  ```

  o bien, mediante la ruta completa a su directorio en /tmpu:

  ```
  cd /tmpu/nombre_de_grupo/nombre_de_usuario
  ```

- **BAUL**
  - Está dedicado a albergar archivos de forma permanente.
  - Es un sistema de archivos tipo NFS y es el segundo espacio de almacenamiento más grande en el equipo de supercómputo Miztli.
  - A diferencia de TMPU, el espacio disponible para cada usuario en BAUL si está limitado.
  - Se recomienda utilizar BAUL para almacenar archivos de resultados y/o archivos que ya no serán accedidos frecuentemente.
  - Sólo se puede usar desde el nodo de `login`. Es decir, no se pueden usar archivos en BAUL en los jobs que se ejecuten mediante el sistema de colas.
  - El espacio disponible en BAUL se reparte entre todos los integrantes de un grupo.

  Para conocer el estado del uso grupal en `$BAUL`, utilice el siguiente comando:

```
% quota -gs alfa_g
Disk quotas for group alfa_g (gid 0000):
Filesystem      blocks quota limit grace files quota limit grace
nfs1_ib:/baul2/ 2908M 97657M 100G        988   0     0
...
```

Para acceder al directorio /baul{N} que le corresponde, utilice la variable de ambiente BAUL:

```
cd $BAUL
```

o bien, mediante la ruta completa a su directorio en `$BAUL`:

```
cd /baulN/nombre_de_grupo/nombre_de_usuario
```

donde N puede ser 1 o 2 dependiendo del grupo.

- **HOME**
  - Es el espacio de almacenamiento más pequeño.
  - Es un sistema de archivos tipo NFS.
  - Es el directorio de inicio al acceder al equipo.
  - Al igual que BAUL, el espacio disponible en HOME está limitado por cuotas del sistema.
  - No debe utilizarse para almacenar archivos temporales generados por sus jobs.

  Para conocer su cuota, utilice el siguiente comando:

```
% quota -s
Disk quotas for user alfa (uid 0000):
Filesystem      blocks quota limit grace files quota limit grace
nfs1_ib:/home/  20496  489M  500M        816   0     0
```

  Para acceder a su directorio en /home, utilice la variable de ambiente HOME:

```
cd $HOME
```

  o bien, mediante la ruta completa a su directorio en /home:

```
cd /home/nombre_de_grupo/nombre_de_usuario
```

Es importante recordarle que, de acuerdo a los Términos y Condiciones de Uso de la Supercomputadora Miztli, la Coordinación de Supercómputo no realiza respaldos de la información de los usuarios, y por lo tanto no es responsable en ningún caso de pérdida total o parcial. Por lo anterior, se le recomienda hacer respaldos periódicos en su infraestructura local.

4. Uso de aplicaciones

En Miztli, el software instalado y configurado por la Coordinación de Supercómputo es clasificado como:

- **Aplicaciones** - Software de un modelo computacional específico.
- **Bibliotecas** - Programas con colecciones de subrutinas que son utilizadas por otros.
- **Compiladores** - Son programas para generar código ejecutable.
- **Herramientas** - Programas para diseñar, desarrollar y analizar aplicaciones.
- **Personal** - Software propio de un usuario (instalado por la coordinación o por el usuario).

En esta sección llamaremos aplicación a cualquier tipo de software de esta clasificación.

La acciones que se pueden realizar sobre las aplicaciones son:

- Listar las aplicaciones disponibles.
- Consultar información de una aplicación.
- Activar o desactivar el ambiente para usar aplicaciones.
- Listar las aplicaciones activas en el ambiente de trabajo.
- Limpieza del ambiente de trabajo.

Y son efectuadas mediante el comando `module`.

4.1. Aplicaciones disponibles

Mediante la opción `avail`[1], se listan las aplicaciones disponibles:

Listar todas la aplicaciones:

```
$ module av
```

Listar solamente la sección de herramientas:

```
$ MODULEPATH=/opt/SC/modulefiles/herramientas module av
```

Listar solamente la sección de bibliotecas:

```
$ MODULEPATH=/opt/SC/modulefiles/bibliotecas/ module av
```

4.2. Consultar información sobre las aplicaciones

Para mostrar una descripción del software o detalles a tomar en cuenta para el uso de la aplicación, utilizar la opción *help*:

```
$ module help nameApp
```

[1] Algunas opciones del comando `module` pueden ser abreviadas, por ejemplo, `avail` como *av*.

La opción *whatis* muestra una breve descripción del módulo de la aplicación que se indique:

```
$ module whatis nameApp
```

Para desplegar la ubicación del archivo de configuración, así como las modificaciones que sufrirá el ambiente cuando la aplicación sea activada, emplear la opción *show*:

```
$ module show nameApp
```

4.3. Activación y desactivación de aplicaciones

No se puede usar una aplicación sin activarla en el ambiente de trabajo, ya que no están definidas las variables de ambiente necesarias y no son accesibles sus comandos:

a) Sin acceso a los comandos de la aplicación:

```
$ commandApp
-bash: commandApp: command not found
```

b) Variables no definidas:

```
$ if [ -n "${NAMEVAR+set}" ]
> then echo -e "\n Variable SI esta definida, nameVar=$NAMEVAR"
> else echo -e "\n Variable NO esta definida, nameVAr=$NAMEVAR"
> fi

Variable NO esta definida, nameVar=
```

4.3.1. Activación

Con las opciones *add* o *load*, se activa la aplicación en el ambiente de trabajo:

```
$ module load nameApp
```

Verificación después de la activación.

a) Comprobación de acceso a utilerías de la aplicación:

```
$ commandActive
Query = "active"

commandApp> quit
```

b) Variables de ambiente existentes:

```
$ if [ -n "${NAMEVAR+set}" ]
> then echo -e "\n Variable SI esta definida, nameVar=$NAMEVAR"
> else echo -e "\n Variable NO esta definida, nameVar=$NAMEVAR"
> fi

Variable SI esta definida, nameVar=Hello_World
```

4.3.2. Desactivación

Con las opciones *rm* o *unload* se desactiva una aplicación:

```
$ module rm nameApp
```

4.4. Listar aplicaciones activas

Para conocer cuáles son las aplicaciones activas en el ambiente de trabajo, se debe usar la opción `list` o *li* del comando **`module`** :

```
$ module li
```

4.5. Limpieza del ambiente de trabajo

En ocasiones, al tener demasiadas aplicaciones activas o cuando se requiere hacer pruebas con un ambiente limpio, es útil hacer un saneamiento del ambiente de trabajo desactivando todas las aplicaciones, este se lleva acabo mediante la opción *purge*:

```
$ module purge
```

5. Compilación de códigos

En Miztli se cuenta con dos suites de compilación:

- Intel - Necesita ser activada antes de poder usarla:
  `$ module load intel`
- GCC - Se puede utilizar sin previa activación en el ambiente de trabajo.

Los lenguajes que son soportados por estas suites son los siguientes:

Lenguaje	Comandos	
	Intel	**GCC**
C	**`icc`**	**`gcc`**
C++	**`icpc`**	**`g++`**
Fortran	**`gfortran`**	**`ifort`**

5.1. Programas de memoria Compartida

Para habilitar el paradigma de programación OpenMP en ambas suites:

	Intel	**GCC**
bandera	*-qopenmp*	*-fopenmp*

5.2. Programas de memoria Distribuida

Para hacer uso del estándar MPI, se debe utilizar la biblioteca Intel MPI:
`$ module load mpi/intel-5.0.2p-044`

Los comandos a utilizar son los siguientes:

Lenguaje	Comando
C	`mpiicc` / `mpicc`
C++	`mpiicpc` / `mpixx`
Fortran	`mpiifort` / `mpif90`

6. Uso del sistema de colas

6.1. Jobs y colas

Para ejecutar programas en Miztli es necesario *enviar jobs* a alguna de las colas del sistema de colas. Para saber qué colas están disponibles, use el comando `bqueues`. Mediante el uso de opciones y argumentos puede indicarse el tipo de información que se requiere, por ejemplo:

1. Mostrar un listado de las colas disponibles y el estado en el que se encuentran:

```
$ bqueues
```

2. Listar solamente las colas a las que un usuario o un grupo tiene acceso (sustituya `alfa` por su propio *login* y/o `alfa_g` por su propio grupo:

```
$ bqueues -u alfa
$ bqueues -u alga_g
```

3. Al indicar el parámetro -l seguido del nombre de la cola puede consultar detalles sobre una cola en particular. Por ejemplo, el tiempo límite de ejecución, los usuarios que tienen acceso a la cola o los nodos a los que la cola tiene acceso:

```
$ bqueues -l q_htc
```

Actualmente, se cuenta con un esquema de 6 colas, que se encargan de manejar los diferentes tipos de recursos de Miztli[2]:

q_htc : Uso de los nodos regulares para los proyectos HTC.

q_hpc : Uso de los nodos regulares para los proyectos HPC.

q_residual : Uso de los nodos regulares que no estén siendo usados por otras colas.

q_gpu : Uso de los nodos con GPUs.

q_32p_256g, q_16p_256g : Uso de los nodos con 256 Gigabytes de RAM.

Para saber de qué tipo es su proyecto, y cuáles son los límites que tiene asignados, puede utilizar el programa `consulta` (sustituya `alfa` por su propio *login*):

[2]Puede ver más información sobre las colas en el Apéndice A.

```
$ consulta alfa
=================
DATOS DEL USUARIO
=================
Cuenta   = alfa
Grupo    = alfa_g
Proyecto = htc
====================
COLAS           CORES
====================
q_htc              96
q_gpu              32
q_16p_256g         32
q_32p_256g         32
q_residual          -
====================
```

6.2. Envío y monitoreo de jobs.

El envío de jobs se realiza a través del comando `bsub`, al cual se le deben indicar una serie de parámetros y la ruta del programa a ejecutar. Los parámetros más importantes son:

-q Cola a utilizar.

-n Número de cores a utilizar.

-oo Nombre del archivo de salida.

-eo Nombre del archivo de errores.

Estos parámetros pueden especificarse en la línea de comandos. Por ejemplo, la siguiente línea envía un job en el que se ejecutará el program `mdrun`, usando 16 cores en la cola `q_htc`:

```
% bsub -q q_htc -n 16 -oo salida -eo error mdrun
```

A su vez, los parámetros pueden escribirse en un *script* (archivo de texto) que se proporciona como entrada a `bsub`. Por ejemplo, si se tiene un archivo llamado `escript` con el siguiente contenido:

```
#BSUB -q q_hpc
#BSUB -n 32
#BSUB -oo salida
#BSUB -eo error

nwchem input.nw
```

entonces la línea de comando:

```
% bsub < escript
```

es equivalente a:

```
% bsub -q q_hpc -n 32 -oo salida -eo error nwchem input.nw
```

Una vez que se envía un job, el sistema de colas contesta con un mensaje como el siguiente:

```
% bsub -q q_hpc -n 32 -oo salida -eo error nwchem input.nw
Job <885413> is submitted to queue <q_hpc>.
```

El número entre llaves es el *identificador del job* (o JOBID), y sirve para hacer consultas y operaciones sobre el job.

Para monitorear el estado del job, debe usarse el comando `bjobs`, seguido del JOBID. Por ejemplo:

```
$ bjobs 885413
JOBID   USER    STAT  QUEUE   FROM_HOST   EXEC_HOST   JOB_NAME    SUBMIT_TIME
885413  alfa    DONE  q_htc   mn328       32*g2_b     nwchem      Ago 13 17:37
```

La columna `STAT` muestra el *estado* del job, el cual debe ser alguno de los siguientes:

`PEND` El job se encuentra en espera de que se liberen los cores suficientes para iniciar su ejecución.

`RUN` El job se encuentra en ejecución.

`USUSP` El job fue suspendido por el usuario.

`SSUSP` El job fue suspendido por el sistema.

`DONE` El job terminó sin error o por límite de tiempo.

`EXIT` El job terminó con error.

Si quiere saber la razón por la que un job esté en espera (PEND), puede agregar la opción `-p` a `bjobs`:

```
% bjobs -p 885412
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME
SUBMIT_TIME
885412  alfa    PEND  q_residual mn328                   omega
Ago 13 17:34
 Job slot limit reached: 371 hosts;
```

Las razones más comunes son:

- "Job slot limit reached". Significa que el equipo no tiene cores libres.
- "User has reached the per-user job slot limit of the queue". Significa que el usuario ha alcanzado el límite de cores permitido en la cola.

Si ejecuta `bjobs` sin ningún parámetro, recibirá el estado de todos sus jobs en espera o en ejecución, mientras que si agrega la opción `-a` se añadirán los jobs que hayan terminado recientemente.

```
$ bjobs
JOBID   USER    STAT  QUEUE   FROM_HOST   EXEC_HOST   JOB_NAME    SUBMIT_TIME
883664  alfa    RUN   q_htc   mn328       16*g1       *_Freq.out  Ago 10 20:06
883665  alfa    PEND  q_htc   mn328                   *_Freq.out  Ago 10 20:06
883666  alfa    PEND  q_htc   mn328                   *_Freq.out  Ago 10 20:06
883667  alfa    PEND  q_htc   mn328                   *_Freq.out  Ago 10 20:06
883668  alfa    PEND  q_htc   mn328                   *_Freq.out  Ago 10 20:06
```

```
$ bjobs -u alfa -a
JOBID   USER    STAT  QUEUE   FROM_HOST   EXEC_HOST   JOB_NAME    SUBMIT_TIME
883664  alfa    RUN   q_htc   mn328       16*g1       *_Freq.out  Ago 10 20:06
```

```
883665   alfa   PEND  q_htc    mn328                 *_Freq.out  Ago 10 20:06
883666   alfa   PEND  q_htc    mn328                 *_Freq.out  Ago 10 20:06
883667   alfa   PEND  q_htc    mn328                 *_Freq.out  Ago 10 20:06
883668   alfa   PEND  q_htc    mn328                 *_Freq.out  Ago 10 20:06
874877   alfa   DONE  q_16p_256g mn328    16*256g    *_Freq.out  Aug 26 10:46
883662   alfa   DONE  q_htc    mn328      16*g1      *_Freq.out  Ago 10 20:06
883663   alfa   DONE  q_htc    mn328      16*g2_b    *_Freq.out  Ago 10 20:06
```

Puede utilizar el programa `uso` para obtener un resumen del número de cores que se están utilizando en su grupo, de modo que verifique las razones por las que su job está en espera:

```
$ uso
=====================
COLA             CORES
=====================
q_htc              192
q_residual         128
=====================
TOTAL              320
GRUPO            alfa_g
=====================
```

Adicionalmente, cuando se tienen trabajos en ejecución, es posible consultar los límites y la cantidad de recursos en uso por cada grupo, mediante:

```
$ blimits
```

6.3. Memoria compartida y memoria distribuida.

La mayoría de los programas paralelos pueden clasificarse como programas de memoria compartida o de memoria distribuida. Es importante que conozca la naturaleza del programa que va a utilizar, de modo que pueda ser enviado de forma correcta al sistema de colas.

Memoria compartida. En todos los programas de memoria compartida se tiene que utilizar la opción:

```
#BSUB -R  "span[hosts=1]"
```

la cual concentra todos los cores del job en un sólo nodo; si no se utiliza esta opción, el sistema de colas podrá asignar al job cores de diferentes nodos, provocando que la aplicación se ejecute de forma inadecuada o con un comportamiento no esperado en una aplicación de memoria compartida.

La forma de ejecutar un programa de memoria compartida depende del tipo de programa. Aquí mostramos dos de los ejemplos más comunes: programas *OpenMP* y el programa *Gaussian*. El número de cores mostrado es el que corresponde al número de cores de los nodos. Si requiere variar este número, puede ver más información en la sección B.

```
#!/bin/bash

### SCRIPT PARA EJECUTAR JOBS DE PROGRAMAS OPENMP

#BSUB -q q_htc
```

```
#BSUB -n 16,20
#BSUB -R "span[hosts=1]"
#BSUB -oo salida
#BSUB -eo error


### LSB_DJOB_NUMPROC ES UNA VARIABLE QUE CONTIENE
### LA CANTIDAD DE CORES
### QUE HAN SIDO RESERVADOS PARA EJECUTAR EL JOB

export OMP_NUM_THREADS=$LSB_DJOB_NUMPROC

echo "Cantidad de threads $OMP_NUM_THREADS"

echo "Ejecucion de programa de OpenMP"
./app_openmp
```

```
#!/bin/bash

### SCRIPT PARA EJECUTAR JOBS DE GAUSSIAN

#BSUB -q  q_hpc
#BSUB -n  16,20
#BSUB -R  "span[hosts=1]"
#BSUB -oo salida
#BSUB -eo error

module load gaussian/G09-E.01
input=test0001.com

### EL ARCHIVO DE ENTRADA DE GAUSSIAN SE MODIFICA
### PARA QUE LA DIRECTIVA
### %nprocshared CONTENGA LA CANTIDAD DE CORES
###  QUE HAN SIDO RESERVADOS PARA EL JOB

sed -i "s/\(%nprocshared=\).*/\1$LSB_DJOB_NUMPROC/g" $input

echo "Ejecucion de Gaussian "
g09 < $input
```

Memoria distribuida. La mayoría de los programas de memoria distribuida están hechos con la bibliotea *MPI*. Este tipo de programas requieren el uso del comando `mpirun` para su ejecución. Debe evitarse el uso de la opción `-R "span[hosts=1]"`.

```
% bsub -q q_htc -n 32 -oo salida -eo error mpirun ./a.out
```

Dado que los nodos regulares están agrupados en tres conjuntos con características diferentes, es posible escoger un tipo particular de nodos para ejecutar un programa, ya sea de memoria compartida o memoria distribuida. Puede encontrar más información en la sección B.

6.4. Consideraciones adicionales para el uso de nodos regulares

Ya ha sido mencionado que la mayor cantidad de cores en Miztli pertenecen a los nodos "regulares", los cuales son administrados mediante las colas `q_htc`, `q_hpc` y `q_residual`, por lo que es importante que determine la mejor forma de aprovecharlos.

Puede mandar tantos jobs como desee a la cola que le corresponde a su proyecto. El sistema de colas ejecutará los jobs hasta alcanzar el límite de su proyecto, y los demás quedarán en espera. Es importante notar que los límites son por grupo, no por usuario.

Si se desea usar simultáneamente más cores de los que permiten los límites, se pueden enviar los jobs adicionales a la cola q_residual. Los jobs en esta cola iniciarán sólo si hay recursos disponibles.

Debe ser precavido en la selección de la cola para sus jobs. En general, deben evitarse los siguientes escenarios:

a) Solicitar cores más allá de su límite en q_htc o q_hpc, si la máquina no está saturada. Esto sólo haría que sus jobs estén en espera siendo que la máquina tiene cores disponibles. Los cores adicionales que desee usar deberían solicitarse en q_residual.

b) Solicitar cores en q_residual si todavía no ha alcanzado su límite en q_htc o q_hpc. Esto es porque los jobs en q_residual siempre tienen el riesgo de ser eliminados por el sistema en cuanto q_htc o q_hpc demanden cores y no los haya disponibles.

Estos escenarios son, en cierta medida, una contraindicación, pero confiamos en que con la práctica pronto encontrará criterios para decidir qué cola usar.

Cabe mencionar que no es necesario que modifique sus scripts para cambiar la cola de destino del job, pues esto se puede modificar desde la línea de comando. Por ejemplo, el script:

```
#!/bin/bash
#BSUB -q  q_htc
#BSUB -oo salida
#BSUB -eo error
#BSUB -n  64

mpirun ./a.out
```

se ejecutará con 64 procesadores en la cola q_htc con la siguiente instrucción:

```
 % bsub < script.lsf
```

pero se ejecutará en la cola q_residual con la siguiente instrucción:

```
 % bsub -q q_residual < script.lsf
```

Evidentemente, es importante que sepa cuántos cores hay libres para decidir qué cola utilizar. Esto puede conseguirse mediante el comando `cores`:

```
$ cores
==================
Cores disponibles
por grupo de nodos
==================
GRUPO      CORES
256g          96
```

```
g1            577
g2_a          106
g2_b           65
gpu            96
==================
```

en donde los grupos g1, g2_a y g2_b son los que constituyen el conjunto de los nodos regulares. El programa cores también proporciona información sobre las características de los nodos de cada grupo:

```
$ cores g1
Grupo g1
   Nodos con 16 procesadores y 64 GB de RAM.

$ cores g2_b
Grupo g2_b
   Nodos en segmento de red b, con 20 procesadores y 128 GB de RAM.
```

7. Consulta del consumo de recursos

La contabilidad del uso de recursos de cada proyecto se actualiza mensualmente y el titular de la cuenta puede consultar el consumo de recursos de su proyecto en:
https://servicios.super.unam.mx/

Para ingresar al sistema deberá proporcionar los siguientes datos:

Usuario
: Es el mismo nombre de usuario de la cuenta titular en Miztli.

Contraseña
: Se encuentra en un archivo llamado .acceso_portal en el directorio $HOME de la cuenta titular.

A. Descripción de las colas

q_htc. Los miembros de proyectos HTC deben correr sus jobs usando la cola q_htc, cuyas características son:

- Solamente los usuarios con proyectos HTC pueden enviar trabajos a esta cola.
- Cada proyecto tiene un límite de cores que puede usar simultáneamente. Este límite está calculado de acuerdo a la cantidad de recursos que tiene asignado el proyecto.
- Se pueden enviar jobs con cualquier número de cores, hasta el límite del proyecto.
- Se puede enviar cualquier cantidad de jobs.
- La duración máxima de los jobs es de 240 horas.
- q_htc tiene la mayor prioridad, así que los jobs en espera y dentro del límite de esta cola son los primeros en ser despachados cuando se liberan cores.
- Los tiempos de espera de jobs dentro del límite de la cola son mínimos.

q_hpc. Los miembros de proyectos HPC deben correr sus jobs usando la cola q_hpc, cuyas características son:

- Solamente los usuarios con proyectos HPC pueden enviar trabajos a esta cola.
- Cada proyecto tiene un límite de cores que puede usar simultáneamente. Este límite se ha establecido de acuerdo al tipo de software que preponderántemente usan los miembros del proyecto.
- Se pueden enviar jobs con cualquier número de cores, hasta el límite del proyecto.
- Se puede enviar cualquier cantidad de jobs.
- La duración máxima de los jobs es de 72 horas.
- Esta cola es la segunda con mayor prioridad. Todos sus jobs en espera son despachados después de los jobs de q_htc y antes que todas las demás colas.
- Los trabajos en esta cola compiten por los recursos en el esquema “fairshare”, es decir, existe una prioridad dinámica que disminuye conforme se usan más recursos.
- El tiempo de espera de los jobs dependerá del estado de la máquina, pero hay que recalcar que Miztli tiene recursos suficientes para ejecutar simultáneamente jobs de al menos 10 proyectos HPC.

q_residual. En periodos en los que Miztli no está saturada, es posible utilizar más cores de los que establecen los límites de las colas q_htc y q_hpc mediante el uso de q_residual, cuyas características son:

- Cualquier usuario puede usar esta cola.
- No tiene límites en el número de cores.
- La duración máxima de los jobs es de 72 horas.
- Es la cola con menor prioridad, de modo que puede despachar sus jobs solamente si no hay jobs por despachar en q_htc o q_hpc.

- Si todos los cores de los nodos regulares están ocupados, y cualquiera de las colas q_htc o q_hpc recibe un job dentro del límite, se eliminarán tantos jobs de q_residual como sea necesario para ejecutar el job recién recibido.

q_gpu. La cola q_gpu es la que se debe utilizar para tener acceso a los nodos con GPUs. Las características de esta cola son:

- Cualquier usuario puede utilizarla.
- Se pueden usar hasta 2 nodos (32 cores y 4 GPUs) simultáneamente.
- La duración máxima de los jobs es de 96 horas.
- Los jobs deben solicitar al menos 16 cores, para garantizar que no haya interferencias en el uso de las GPUs.

q_32p_256g y q_16p_256g. Estas son las colas que deben utilizarse para tener acceso a los nodos con 256 Gigabytes de RAM. Sus características son:

- Cualquier usuario puede utilizarlas.
- El límite de cores por job es de 16 (16p) o 32 (32p).
- El límite de cores por grupo en la cola es de 32 en ambos casos.
- La duración máxima de los jobs es de 124 horas (16p) o de 72 horas (32p).
- Los jobs deben solicitar al menos 16 cores, para garantizar que no haya interferencias en el uso de la memoria.

B. Más información sobre jobs paralelos

B.1. Descripción de los nodos y sus diferencias

En Miztli, además de que puede utilizar al menos dos colas para el uso de los nodos regulares, éstos se dividen en diferentes tipos de nodos. Estas diferencias consisten tanto en su tipo de procesador, cantidad de cores y cantidad de memoria, como en su conexión a la red de datos.

La Fig. 1 muestra un esquema de la red de datos (infiniband) de Miztli, en el que se puede apreciar los diferentes tipos de nodos regulares y la forma en que están interconectados.

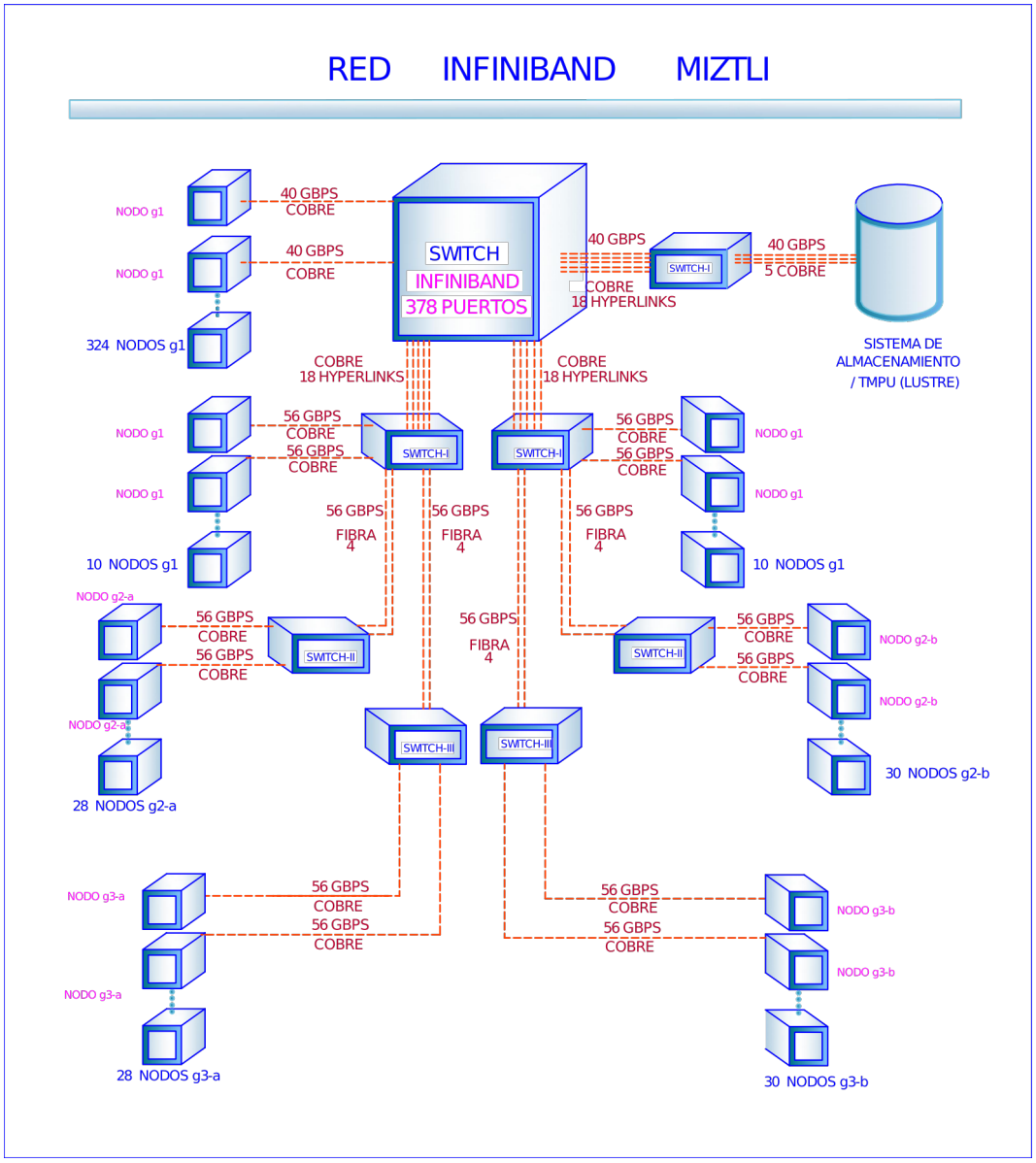

Figura 1: Esquema de la red Infiniband de Miztli.

Los tipos de nodos regulares son:

- Nodos g1. Son los nodos regulares originales de Miztli. Sus procesadores son Intel Xeon E5-2670 v1. Cada nodo tiene 16 cores y 64 Gigabytes de RAM. Son 314 nodos.
- Nodos g2_a y g2_b. Son los nodos añadidos en 2016. Sus procesadores son Intel E5-2660 v3. Cada nodo tiene 20 cores y 128 Gigabytes de RAM. Son 28 y 30 nodos en cada grupo, respectivamente.
- Nodos g3_a y g3_b. Son los nodos añadidos en 2017. Sus procesadores son Intel E5-2683 v4. Cada nodo tiene 32 cores y 256 Gigabytes de RAM. Son 28 y 30 nodos en cada grupo, respectivamente.

Los nodos g1 están conectados al Switch de 378 puertos, ya sea de forma directa o a través de

un switch adicional (switch tipo I) que se conecta a través de 18 *hyperlinks*. La conectividad entre todos estos nodos es *simétrica*: el ancho de banda es el mismo entre cualquier par de nodos, y su valor nominal es de 40 Gbps.

Los nodos g2_a están conectados entre sí mediante un switch tipo II y al resto de los nodos mediante 4 cables de fibra que van a un switch tipo I. Los 28 nodos que hay en el grupo tienen que repartirse el ancho de banda que proporcionan estas cuatro conexiones. De este modo, la conectividad de los nodos g2_a es *asimétrica*: entre nodos del mismo grupo su valor nominal es de 56 Gbps, pero hacia nodos de otros grupos es de 8 Gbps. Lo mismo se puede decir de los 30 nodos del grupo g2_b, así como de los 28 nodos del grupo g3_a y los 30 del grupo g3_b.

Ejecutar un programa paralelo que utilice nodos de diferentes grupos sería ineficiente, dada la asimetría de g2_a y g2_b. Por esta razón, el sistema de colas de Miztli *está configurado para proporcionar todos los cores que se soliciten en un job solamente con nodos de un mismo grupo*. Esta es una restricción para cada job, pero el usuario no debe preocuparse por seleccionar el tipo de nodos, pues esta tarea la hace de forma automática el sistema de colas.

En términos generales, solamente debe preocuparse por especificar el número de cores de su job, sin embargo, la siguiente información le puede ser útil para aprovechar mejor sus recursos.

B.2. Otros casos de jobs de memoria compartida

Un nodo parcial. Como ya se mencionó, existen nodos tanto de 16 como de 20 cores. Si desea utilizar una cantidad de cores menor a 16, su job puede ser ejecutado en cualquier tipo de nodo. En este caso, puede usar como referencia los siguientes scripts:

```
#!/bin/bash

#!/bin/bash

### SCRIPT PARA EJECUTAR JOBS DE PROGRAMAS OPENMP

#BSUB -q q_htc
#BSUB -n 8
#BSUB -R "span[hosts=1]"
#BSUB -oo salida
#BSUB -eo error


### LSB_DJOB_NUMPROC ES UNA VARIABLE QUE CONTIENE
### LA CANTIDAD DE CORES
### QUE HAN SIDO RESERVADOS PARA EJECUTAR EL JOB

export OMP_NUM_THREADS=$LSB_DJOB_NUMPROC

echo "Cantidad de threads $OMP_NUM_THREADS"

echo "Ejecucion de programa de OpenMP"
./app_openmp
```

```
#!/bin/bash

### SCRIPT PARA EJECUTAR JOBS DE GAUSSIAN
```

```
#BSUB -q  q_hpc
#BSUB -n  8
#BSUB -R  "span[hosts=1]"
#BSUB -oo salida
#BSUB -eo error

module load gaussian/G09-E.01
input=test0001.com

### EL ARCHIVO DE ENTRADA DE GAUSSIAN SE MODIFICA
### PARA QUE LA DIRECTIVA
### %nprocshared CONTENGA LA CANTIDAD DE CORES
###  QUE HAN SIDO RESERVADOS PARA EL JOB

sed -i "s/\(%nprocshared=\).*/\1$LSB_DJOB_NUMPROC/g" $input

echo "Ejecucion de Gaussian "
g09 < $input
```

Usar todos los cores de un nodo. Si no le importa qué tipo de nodo usar, pero requiere usar todos los cores de un nodo, es necesario indicarle al sistema de colas que el número de cores solicitado depende del tipo de nodo asignado. Esto se puede realizar de la siguiente forma:

```
#!/bin/bash
#BSUB -q q_hpc

### LA SIGUIENTE OPCION INDICA QUE SE PUEDEN USAR 16 O 20 CORES

#BSUB -n 16,20
#BSUB -R "span[hosts=1]"
#BSUB -oo salida
#BSUB -eo error

export OMP_NUM_THREADS=$LSB_DJOB_NUMPROC
echo "Cantidad de threads $OMP_NUM_THREADS"
echo "Ejecucion de programa de OpenMP"
./app_openmp
```

El sistema de colas buscará un nodo de 20 cores que esté completamente desocupado, y se lo asignará al job. Si no existe ninguno, buscará un nodo de 16 cores y se lo asignará al job.

Un tipo de nodo en específico. Si requiere usar un tipo de nodo en específico, puede usar algunos de los siguientes ejemplos como base:

```
#!/bin/bash
#BSUB -q q_htc


#BSUB -n 20
```

```
### LA SIGUIENTE OPCION INDICA QUE EL NODO A USAR
### DEBE SER DE 20 CORES

#BSUB -m "g2_a g2_b"
#BSUB -R "span[hosts=1]"
#BSUB -oo salida
#BSUB -eo error

export OMP_NUM_THREADS=$LSB_DJOB_NUMPROC
echo "Cantidad de threads $OMP_NUM_THREADS"
echo "Ejecucion de programa de OpenMP"
./app_openmp
```

```
#!/bin/bash
#BSUB -q pruebas_2g

#BSUB -n 16
### LA SIGUIENTE OPCION INDICA QUE EL NODO A USAR
### DEBE SER DE 16 CORES

#BSUB -m "g1"

#BSUB -R "span[hosts=1]"
#BSUB -oo salida
#BSUB -eo error

export OMP_NUM_THREADS=$LSB_DJOB_NUMPROC
echo "Cantidad de threads $OMP_NUM_THREADS"
echo "Ejecucion de programa de OpenMP"
./app_openmp
```

B.3. Otros casos de jobs de memoria distribuida

1. A cualquier tipo de nodo, con cualquier distribución de nodos.

   a) Forma tradicional.

   ```
   $ bsub -n NC < script.ls
   ```

   El trabajo iniciará si hay NC cores disponibles en cualquier grupo, sin importar la forma en que serán repartidos los procesos entre los nodos asignados.

   b) Número de cores variable.

   ```
   $ bsub -n nmin,nmax < script.ls
   ```

   El trabajo iniciará si al menos hay *nmin* cores disponibles.

2. A un tipo de nodo, con una prioridad específica.

   a) Preferencia de un tipo de nodo.

   ```
   $ bsub -n NC -m "g2_a+2 g1+3 g2_b+1" < script.ls
   ```

El trabajo comprobará si hay NC cores disponibles en el grupo $g1$ e iniciará si esta condición se cumple, de lo contrario realizará la misma verificación para $g2\_a$ y $g2\_b$, en el orden de prioridad especificado.

3. A un tipo de nodo específico.

   a) Eligiendo un tipo de nodo

```
$ bsub -n NC -m "g2_b" < script.ls
```

   El trabajo iniciará si hay NC cores disponibles en un nodo del tipo $g2\_b$.

Créditos

Este es un material técnico realizado como guía base para el uso de la supercomputadora Miztli, con la colaboración de:

Silvia Elizabeth Frausto Del Río.

Leobardo Itehua Rico.

Irving Carlos Álvarez Castillo.

José Luis Gordillo Ruiz.

Camilo González González. Elaboración de la Figura 1. "Esquema de la Red Infiniband de Miztli".

Contactos

José Luis Gordillo Ruiz
Coordinador de Supercómputo, DGTIC-UNAM
Correo: jlgr@super.unam.mx
Teléfono: 5622-8529

Irving Carlos Álvarez Castillo
Paralelismo y Optimización
Coordinación de Supercómputo, DGTIC-UNAM
Correo: irving@super.unam.mx
Teléfono: 5622-8599

Leobardo Itehua Rico
Paralelismo y Optimización
Coordinación de Supercómputo, DGTIC-UNAM
Correo: itehua@super.unam.mx

Teléfono: 5622-8164

Silvia Elizabeth Frausto Del Río
Administración de Supercómputo
Coordinación de Supercómputo, DGTIC-UNAM
Correo: silvia@super.unam.mx
Teléfono: 5622-8599