Cómo usar Matlab 2023a en el HPC de la Universidad de Cádiz

Introducción

La Universidad de Cádiz dispone actualmente de una licencia campus para usar el software Matlab. En virtud de esta licencia se ha instalado dicho software en el cluster HPC en su versión 2023a con el conjunto completo de herramientas que dicho software suministra.
El servidor de licencias dispensa tanto la licencia *«MATLAB Parallel Server»* como la versión *«MATLAB (Concurrent)»*
Para poder usar MATLAB debemos cumplir con los siguientes pasos:

- Tener instalado en nuestro equipo MATLAB en su versión independiente en el equipo del investigador (equipo local). Debe asegurarse tener instalada la herramienta *«Parallel Computing Toolbox»*. Este paso se considera requisito y no está cubierto en este documento.
- Configurar nuestra cuenta en el cluster HPC.
- Añadir los scripts necesarios a la instalación en nuestro equipo local.
- Generar el perfil del cluster.

El documento finaliza con la configuración de trabajos para ser lanzados en el cluster.

Configuración de la cuenta en el cluster

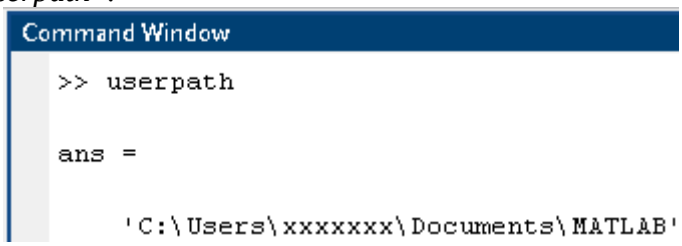
La configuración de nuestra cuenta en el cluster es muy simple: es sólo asegurarse de que las variables necesarias para ejecutar MATLAB están presentes en el momento de iniciar sesión, sin necesidad de hacer nada más.
Para ello basta con añadir la línea que carga el módulo correspondiente a nuestro fichero .bashrc.
Para ello iniciamos sesión ssh en el cluster y ejecutamos la orden:

```
09:57:29 uXXXXXXXX@urania01.uca.es:~$ echo -e "module load Matlab/R2023a\n" >> ~/.bashrc
09:58:03 uXXXXXXXX@urania01.uca.es:~$ tail -2 .bashrc
module load Matlab/R2023a

09:58:13 uXXXXXXXX@urania01.uca.es:~$
```

(`echo -e "module load Matlab/R2023a\n" >> ~/.bashrc`)
NOTA: La segunda orden es para comprobar que la línea se ha añadido al fichero correctamente.

Añadir los scripts necesarios a nuestro equipo local

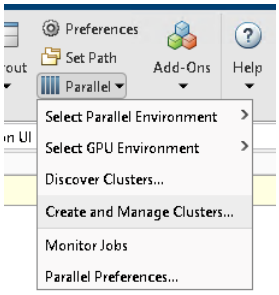
Lo primero es descargarnos los scripts necesarios desde la página web de supercomputación: https://supercomputacion.uca.es/wp-content/uploads/2023/06/Universidad-de-Cadiz.Desktop.zip
Para saber dónde descomprimir este archivo, en nuestro equipo local, iniciamos MATLAB y ejecutamos la orden *«userpath»*:

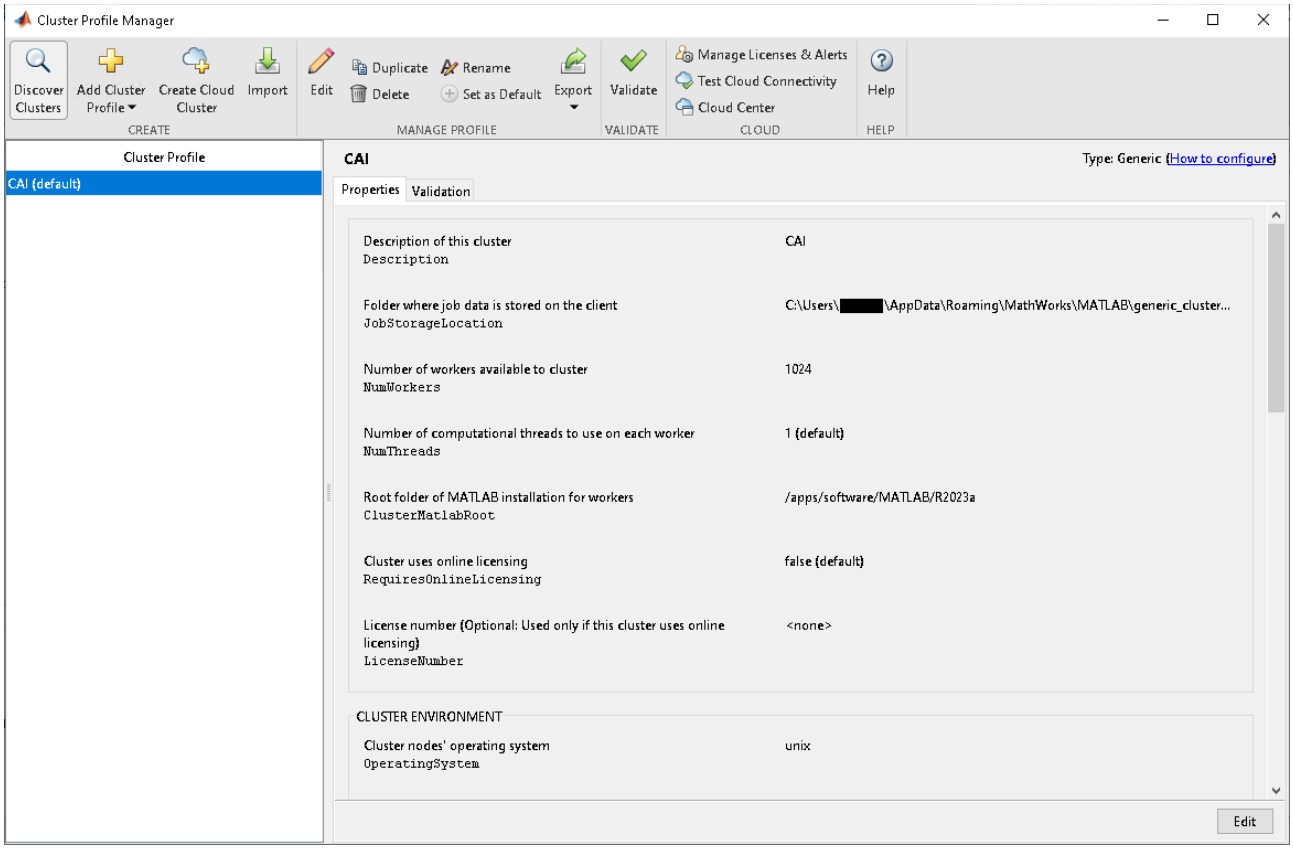


Descomprimimos el archivo en esa localización.

Generar el perfil del cluster

Ahora debemos generar el perfil de nuestro cluster. Para ello ejecutamos la orden *«configCluster»*. Este script está personalizado para nuestra Universidad, por lo que lo único que pedirá será nuestro nombre de usuario en el cluster (*«uDNI»*).
Una vez hecho esto, en la lista de perfiles de clusters de supercomputación aparecerá el perfil *«CAI»* como perfil por defecto: pinchamos en *«Parallel»* → *«Create and Manage Clusters...»* y obtenemos:



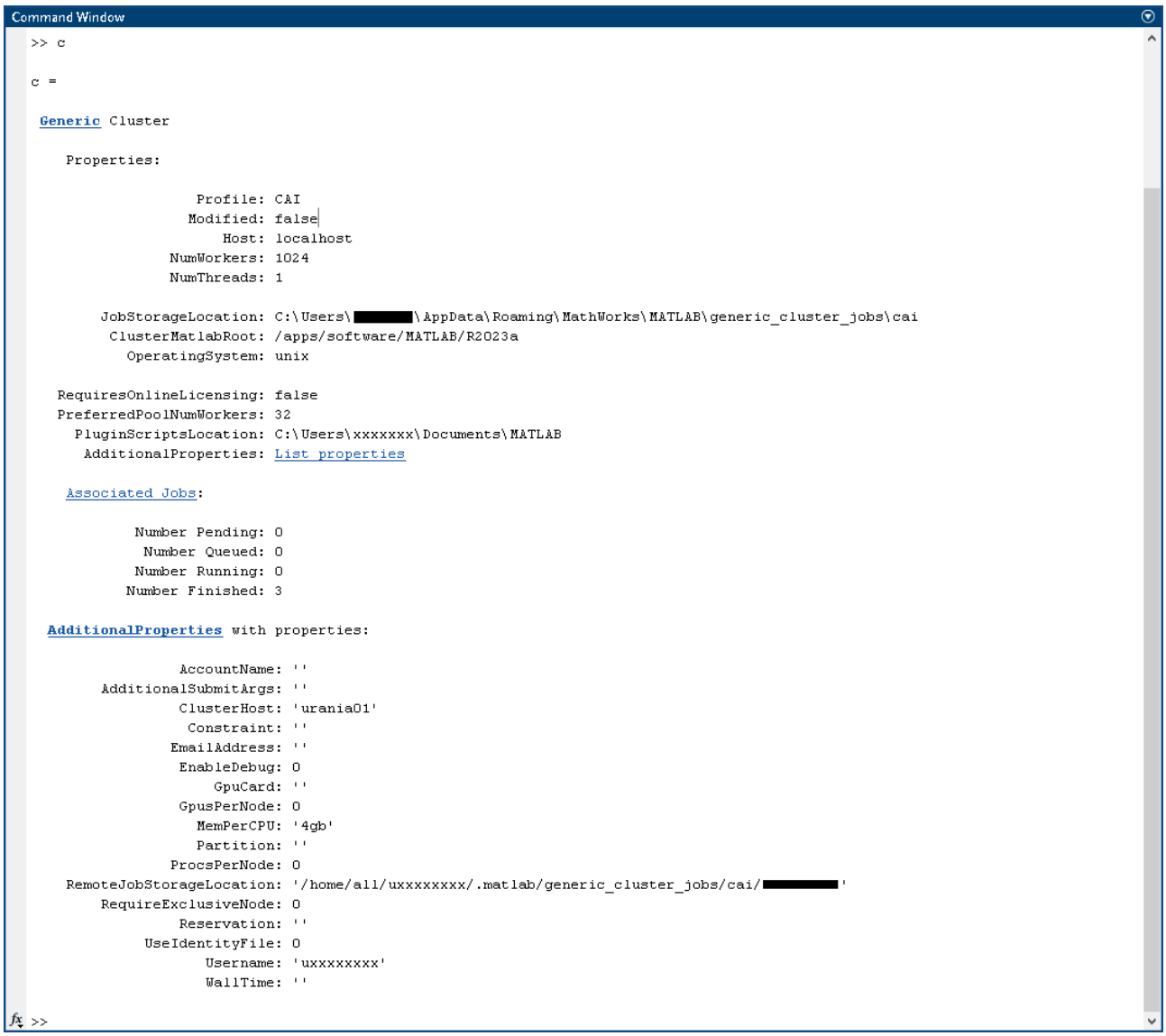


Configuración de los trabajos

Antes de enviar los trabajos (*«submit»*) podemos especificar varios parámetros para pasárselos a nuestros trabajos como la cola, el email para avisarnos, la duración esperada, etc.
Lo primero es obtener un manejador para el cluster:

```
>> c = parcluster;
```

Aquí si ejecutamos «c» y pinchamos en «list properties» veremos todo lo que podemos configurar para nuestros trabajos:



Para definir, por ejemplo, una duración de 3 días y 6 horas haríamos:

```
>> c.AdditionalProperties.WallTime = '3-06:00';
```

Para especificar que vamos a usar dos gpus debemos hacer:

```
>> c.AdditionalProperties.GpusPerNode = 2;
>> c.AdditionalProperties.Partition = 'gpu';
```

La propiedad «*AdditionalSubmitArgs*» sirve para añadir cualquier opción no especificada aquí.

Si queremos que estas modificaciones estén disponibles entre diferentes sesiones MATLAB, nuestros valores por defecto, debemos salvar las modificaciones con:

```
>> c.saveProfile;
```

Lanzar un trabajo

Para lanzar un trabajo usamos el método «*batch*» de parcluster, que devuelve un objeto del tipo trabajo que debemos guardar para poder acceder a los resultados:

```
>> job = c.batch(función,N,X1,X2,...);
```

(**NOTAS**:

- El script «función» es copiado al nodo que ejecutará el trabajo. No hay que incluir la extensión «.m».
- N es el número de argumentos devueltos por el script.
- X_1, X_2, … son los argumentos de entrada al script.

)

Con esta acción el trabajo se está ejecutando en el cluster y podemos cerrar nuestro equipo.

Para reanudar el trabajo debemos volver a obtener un manejador para el cluster y solicitar la lista de trabajos en ejecución:

```
>> c = parcluster;
```

```
>> jobs = c.Jobs;
```
Si tenemos varios trabajos en ejecución y queremos comprobar el número dos podemos hacer:
```
>> job = c.Jobs(2);
>> job.State
```
Que nos dirá si está en ejecución o ha terminado (*«finished»*). Para obtener los resultados, si pasamos una función (*«@pwd»* por ejemplo) usamos *«fetchOutputs»* y si pasamos un script usamos *«load»*.
```
>> job.fetchOutputs{:}
>> load(job,'x');
```
Finalmente eliminamos el trabajo para liberar los recursos:
```
>> job.delete;
```

Lanzar un trabajo paralelo

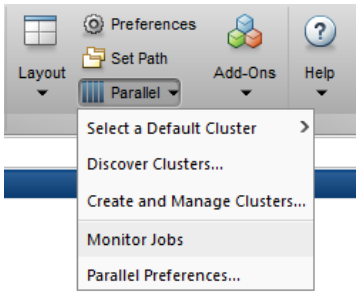
Vamos a suponer este script:

```
function [t, A] = parallel_example(iter)

if nargin==0
    iter = 8;
end

disp('Start sim')

t0 = tic;
parfor idx = 1:iter
    A(idx) = idx;
    pause(2)
    idx
end
t = toc(t0);

disp('Sim completed')

save RESULTS A

end
```

Esta vez, al usar batch especificamos un *«pool»* (en este caso será de cuatro nodos):
```
>> job = c.batch(@parallel_example, 1, {16}, 'Pool',4, ...);
```
Por lo demás el trabajo es igual al anterior.

Monitorizar los trabajos

Alternativamente a la expresión *«job.State»* podemos usar la interfaz gráfica para ver el estado de nuestros trabajos:



Ahí podemos ver su estado e incluso eliminarlos pinchando con el botón derecho del ratón sobre el trabajo y seleccionando *«Delete»*.

Depuración de errores

Para ver si han ocurrido errores podemos obtener lo que hubiera sido la salida por pantalla con el comando:

```
>> diary(job)
```

Si queremos ver la salida de depuración usamos la expresión:

```
>> c.getDebugLog(job)
```

Por último, puede ser interesante saber el ID del trabajo en el cluster. Se obtiene con la expresión:

```
>> schedID(job)
```

Una vez obtenido ese número se pueden usar los comandos apropiados en el cluster para obtener información.

Más información

- Parallel Computing Coding Examples
- Parallel Computing Documentation
- Parallel Computing Overview
- Parallel Computing Tutorials
- Parallel Computing Videos
- Parallel Computing Webinars