Introducción al AJO

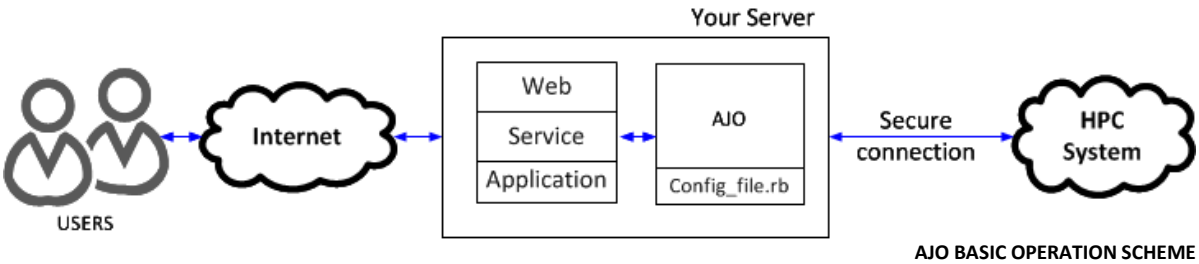
Mayo 2014

Contenidos

1. Introducción

AJO es un acrónimo formado por las iniciales de Asynchronous Job Operator (operador de trabajos asíncrono). Esta herramienta ha sido diseñada y desarrollada con el objetivo de proporcionar una puerta de enlace transparente entre una web, aplicación o servicio y un sistema de alto rendimiento (HPC)

AJO traslada las ejecuciones a un sistema de colas compatible con DRMAA (Distributed Resource Management Application API), como la familia Grid Engine o TORQUE, permitiendo el envío, ejecución y recuperación de cualquier tarea de forma simple y rápida.



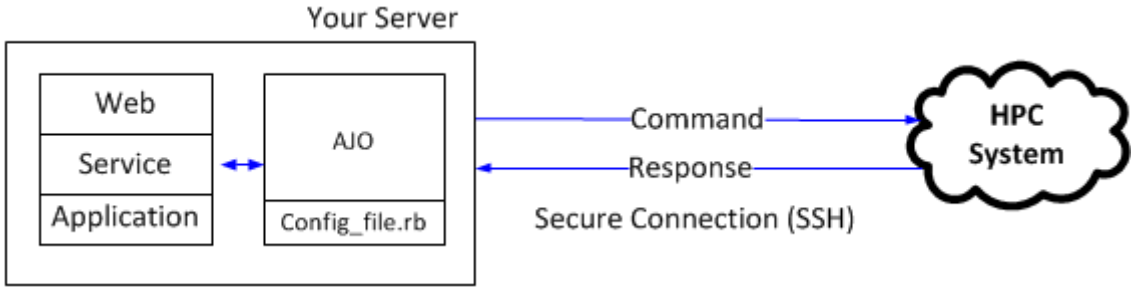
AJO BASIC OPERATION SCHEME

Toda la información sobre el proyecto AJO se encuentra disponible en http://rdlab.lsi.upc.edu/ajo. Para más información contacte con rdlab@lsi.upc.edu.

1.1 Funcionalidades de AJO

AJO proporciona tres funcionalidades principales que permiten controlar el proceso de ejecución al completo.

- SUBMIT: Envía un trabajo al sistema HPC y devuelve un TOKEN ID generado de forma segura.
- QUERY: Comprueba el estado de un trabajo enviado previamente.
- RETRIEVE: Obtiene el resultado de un trabajo terminado.



Además, ofrece algunas funcionalidades extra que completan las anteriores. En el capítulo "Ejecutando y testeando AJO" se recogen los detalles para todas las opciones.

- LIST: Muestra una lista de identificadores los trabajos enviados mediante AJO.
- CANCEL: Cancela un trabajo en ejecución enviado mediante AJO.
- ERASE: Borra la información en el servidor de un trabajo enviado mediante AJO.

2. Instalación

Para ejecutar AJO se requiere un sistema GNU/Linux con Ruby 1.9.1 o superior instalado y un cliente SSH. También es necesario un sistema de colas compatible en ejecución, como puede ser uno de la familia Grid Engine.

La versión más reciente de AJO se encuentra disponible en nuestro repositorio SVN usando "public_ajo" como usuario y contraseña.

```
svn co http://svn-rdlab.lsi.upc.edu/subversion/ajo/public --username public_ajo
```

3. Configurando AJO

Atención: Todas las tareas ejecutadas por AJO se almacenan por defecto en el *home* ($HOME/.executions) de usuario del sistema HPC.

3.1 Opciones generales

En el directorio de descarga de AJO se encuentran diversos archivos, incluyendo un ejemplo de archivo de configuración de nombre *config.rb.* Este script contiene todas las opciones necesarias para correr AJO correctamente.

La primera parte permite especificar opciones básicas tales como a qué servidor conectarse o el usuario a usar, así como el directorio donde se guardará información en el servidor o la ubicación del binario SSH.

```
# SSH options
SERVER = "server.name.domain"
USER = "username" # insert your Linux username between quotes
SSH_CMD = "/usr/bin/ssh #{USER}@#{SERVER}"
AJO_DIR = `#{SSH_CMD} 'echo $HOME'`.chomp("\n") + "/.executions"
```

Por otra parte, AJO envía comandos SSH directamente al clúster, por lo que es necesario usar SSH sin contraseña mediante un par de claves pública y privada. Más información sobre el uso de SSH sin contraseña se halla disponible en la red: http://www.debian-administration.org/articles/152

Si AJO se ejecutará por un servicio del sistema con su propio usuario (por ejemplo Apache o Tomcat) las claves han de ponerse en el $HOME/.ssh del usuario que ejecuta estos servicios.

Además, si este usuario establecerá conexión al servidor (*server*) con más de un usuario (*user1, user2*), se requiere de un archivo *config* en su $HOME/.ssh. La clave pública puede concatenarse en un solo archivo (id_rsa.pub), pero las claves privadas han de hallarse en archivos diferentes, de modo que el archivo *config* especificará qué clave corresponde a cada usuario.

A continuación se muestra un ejemplo de cómo quedaría el archivo, asumiendo que /var/www es el $HOME de Apache (www-data):

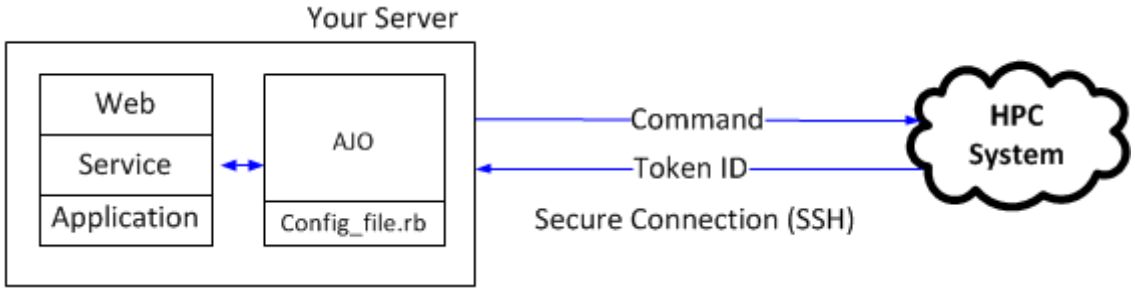
```
root@machine: cat /var/www/.ssh/config
      Host server
      HostName server.domain.com
      IdentityFile ~/.ssh/id_rsa.user1
      User    user1

      Host server
      HostName server.domain.com
      IdentityFile ~/.ssh/id_rsa.user2
      User    user2
```

3.2 Grid Engine y opciones de codificación

En esta sección se encuentran las principales opciones relativas al SGE como el *path* a los binarios, la arquitectura, etc. Se recomienda no modificar esta sección a no ser que sea estrictamente necesario. A continuación se muestra un ejemplo:

```
# SGE options, normally you will not have to change this.
SGE_ROOT = `#{SSH_CMD} 'echo $SGE_ROOT'`.chomp "\n"
SGE_ARCH = `#{SSH_CMD} '#{SGE_ROOT}/util/arch'`.chomp "\n"
SGE_UTIL_PATH = SGE_ROOT + "/bin/" + SGE_ARCH
QSUB_CMD = SGE_UTIL_PATH + "/qsub"
QSTAT_CMD = SGE_UTIL_PATH + "/qstat"
QACCT_CMD = SGE_UTIL_PATH + "/qacct"
QDEL_CMD = SGE_UTIL_PATH + "/qdel"
```

Además, AJO usa un sistema de codificación para crear un *Token ID* seguro para cada envío de trabajo para asegurar la fiabilidad entre la aplicación y el sistema HPC. Esta sección contiene los parámetros usados para codificar la información.



Se recomienda cambiar el valor de la *sal* por algo aleatorio, así como las constantes USER y SERVER en las líneas siguientes por algo más seguro, como las cadenas “ajksdbh8os7dtfg8wky” o “I5787s4wa%^W/3w6u43” para una mejor codificación. Es importante usar comillas para envolver las cadenas.

```
# Encryption options, replace USER and SERVER in second and third lines with
# something random for more security. But be careful - loosing CIPHER_KEY
# and CIPHER_IV will make decoding your job identifier impossible.
CIPHER_SALT = "ajo"
CIPHER_KEY = OpenSSL::PKCS5.pbkdf2_hmac_sha1(USER, CIPHER_SALT, 2000, 16)
CIPHER_IV = OpenSSL::PKCS5.pbkdf2_hmac_sha1(SERVER, CIPHER_SALT, 2000, 16)
```

3.3 Parámetros de carpeta y archivo

AJO empaqueta todos los datos locales y envía una copia al sistema HPC (a $HOME/.executions por defecto), de modo que los archivos de entrada y los directorios deben ser definidos en el archivo de configuración. Además, se deben definir también los archivos de salida donde recuperar los datos después de la ejecución.

En el archivo de configuración, después de la sección de opciones generales, se encuentran los bloques de parámetros de carpeta y archivo. Se trata de *hashes* clave-valor en Ruby. Se debe tener en cuenta que la clave es un símbolo (un tipo especial de variable que empieza por dos puntos (":")) y que el valor debe ser una cadena de caracteres. Se pueden añadir tantos pares clave-valor como sea necesario. Los pares con valor vacío serán ignorados.

El primer bloque, FOLDER_ARGS, permite especificar los parámetros relativos a las carpetas que van a ser usadas por el comando enviado al clúster. Es posible usar *paths* relativos y atajos como '~'. Las variables Bash de sistema también son aceptadas. A continuación se muestra un ejemplo del uso de este parámetro:

```
FOLDER_ARGS = {
    :fld1 => "~/folder42",
    :fld2 => "",
    :fld3 => "",
}
```

El siguiente parámetro, FILE_ARGS, permite seleccionar los archivos que van a ser usados posteriormente en el comando. El formato es el mismo que en el bloque previo:

```
FILE_ARGS = {
    :file1 => "~/file42",
    :file2 => "",
    :file3 => "",
}
```

El siguiente bloque muestra las variables de salida (*output*). FOLDER_OUTPUT y FILE_OUPUT tienen el mismo formato y función que los parámetros descritos anteriormente, pero éstos estarán disponibles posteriormente para una fácil recuperación, ya que será posible descargarlos rápidamente una vez la ejecución del trabajo haya terminado. Su aspecto es el siguiente:

```
FOLDER_OUTPUT = {
     :fld1 => "outputfolder42",
     :fld2 => "",
     :fld3 => "",
}

FILE_OUTPUT = {
     :file1 => "file1.txt",
}
```

El último parámetro es RETRIEVE. Éste permite especificar las variables de salida que se desea recuperar tras una ejecución. El parámetro se define del siguiente modo:

```
RETRIEVE = {
     :folder1 => "file1.txt",
}
```

Finalmente, encontramos el bloque donde se especifica el comando. En éste se pueden escribir los comandos que se desean ejecutar en el clúster. El formato de los parámetros de archivos a usar aquí es ligeramente complejo:

- En primer lugar, las variables configuradas en los bloques superiores son accedidas en minúsculas para el correcto proceso de los *path* antes de enviar el trabajo.
- En segundo lugar, los comandos son cadenas de caracteres. Se recomienda no usar dobles comillas (“) dentro de los comandos o bien escaparlas (mediante \”).

Para usar los argumentos previamente configurados dentro del comando es necesario usar interpolación (*string interpolation*). En Ruby se realiza escribiendo `#{variable}` (almohadilla seguido del nombre de la variable entre llaves).
Por ejemplo, si se quiere mostrar el contenido de `‘file1’` de `‘FILE_ARGS’`, es necesario acceder al nombre del parámetro en minúsculas, en este caso `‘file_args’`, y a continuación acceder al índice específico en este parámetro con el nombre entre corchetes.

De este modo, el comando quedaría de la siguiente manera:

```
“cat #{file_args[:file1]}”.
```

Cualquier variable previamente configurada puede ser usada. A continuación se muestra un ejemplo del bloque de comando:

```
def process_commands file_args, folder_args, file_output,
folder_output, input_dir, output_dir
[
     "uname -a > #{file_output[:file1]}",
```

```
    "echo 'hello' > #{file_output[:file1]}"
]
end
```

4. Ejecutando y testeando AJO

AJO se ejecuta por línea de comandos. Estas son las opciones disponibles:

`--api`
: Pide a AJO que imprima solo información práctica. Útil si la salida va a ser analizada.

`-c FILE or --config FILE`
: Por cada acción, AJO necesita un archivo de configuración, ya que utiliza las opciones SSH que contiene para la mayoría de operaciones. Por defecto busca un archivo de configuración de nombre `config.rb` en el directorio donde se ejecuta a menos que se invoque AJO con la opción `-c FILE`, donde FILE es un archivo de configuración distinto.

`-d DIR or --retrieve-directory DIR`
: Descarga el resultado de la ejecución a DIR.

`--download-exec-folder ID`
: Recupera la carpeta donde se ha ejecutado el trabajo en el sistema HPC.

`-e ID or --erase`
: Borra el directorio asociado al trabajo en el servidor.

`-l or –list`
: Lista todos los identificadores disponibles para consulta y recuperación de datos.

`--library FILE`
: Especifica la ubicación de la librería *libhpc*.

`--log-all`
: Fuerza AJO a hacer *log* de cada paso que realiza al ejecutar una tarea.

  Por defecto, AJO crea un archivo de log en el directorio donde se ejecuta llamado AJO.log y escribe en él solamente los mensajes de error. El archivo se sobrescribe a diario. En caso de ocurrir algún problema con AJO, se recomienda ejecutar el comando con el parámetro --log-all para obtener más información.

`-q ID or --query ID`
: Obtiene la información sobre el estado del trabajo con identificador ID.

`--qstat ID`

Obtiene la salida de la llamada qstat respecto al trabajo con identificador ID.

`--qstat-xml ID`
Obtiene la salida XML de la llamada qstat respecto al trabajo con identificador ID.

`-r ID or --retrieve ID`
Descarga los resultados de la ejecución del trabajo especificado por ID.

`-s or --submit`
Envía al clúster el trabajo especificado en el archivo de configuración.

`-v or –version`
Muestra por pantalla la versión de AJO y termina.

`-x ID or --cancel ID`
Cancela un trabajo en ejecución.

4.1 Ejemplos

Las operaciones estándar con AJO incluyen el envío de trabajos, la consulta del estado, la obtención de resultados y el borrado de los archivos del trabajo en el clúster.

Para enviar el trabajo es necesario ejecutar el siguiente comando:

`AJO -c config.rb -s`
Este comando indicará a AJO que el archivo a usar es config.rb, ubicado en el mismo directorio desde el que ejecutamos, y que deseamos enviar al clúster el trabajo descrito por éste. El comando devolverá un identificador que deberá ser usado posteriormente para otras operaciones.

Para consultar el estado:

`AJO -c config.rb -q identifier_here`
El comando devolverá un mensaje describiendo el estado del trabajo.

Para obtener los resultados:

`AJO -c config.rb -r identifier_here`
También es posible especificar, mediante la opción `-d directory`, el directorio en el que se desea almacenar los datos descargados. Asimismo, es posible pedir la descarga de toda la salida mediante el *flag* `-a`.

Para borrar del clúster los archivos del trabajo:

`AJO -c config.rb -e identifier_here`

Para mostrar los identificadores disponibles para recuperación:

`AJO -c config.rb –l`

Ejecutando este comando se puede, después de configurar AJO, comprobar que todo funciona correctamente. Si no hay trabajos enviados anteriormente mediante AJO, la salida será una línea vacía. Si aparece un error, es recomendable comprobar la sintaxis en el archivo de configuración (puede ser por falta de comas, punto y comas, etc.), actualizar las dependencias de Ruby, verificar la conexión de red y verificar la conexión SSH.