

MANUAL DE GRIDGAIN

Fernando Sierra Pajuelo - NandEx -

1 de septiembre de 2008

Índice

1. Manual de GridGain	2
1.1. Introducción	2
1.2. ¿Qué es GridGain?	2
1.3. Instalación	2
1.4. Ejemplo HolaMundo	4
1.5. Gridificando nuestra aplicación	8

1. Manual de GridGain

1.1. Introducción

Este manual pretende ser una guía de ayuda práctica para todos aquellos que se quieren iniciar en Gridgain y no tienen muy claro por donde empezar. Se presentan ejemplos para entender mejor la forma de utilizar Gridgain.

1.2. ¿Qué es GridGain?

GridGain es una aplicación que permite gridificar programas, principalmente creados en lenguaje Java, ayudando al programador a incrementar la productividad de sus trabajos. El fin es utilizar la tecnología Grid con el propósito de paralelizar los procesos y se conduce a una mejor escalabilidad y rendimiento. Muchos usuarios crean sus primeros ejemplos en menos de 30 minutos, siendo este el tiempo que se tarda en descargar e instalar otros entornos de computación grid. Para más información www.gridgain.com

1.3. Instalación

Lo primero es tener instalado en nuestro Sistema Operativo Java SE Development Kit (JDK) y el compilador que se va a usar, en nuestro caso NetBeans 6.1. A continuación descargamos la versión de GridGain para nuestro Sistema Operativo y realizamos la instalación. Se deben crear las variables de entorno JAVA_HOME y GRIDGAIN_HOME.

- Windows XP
 1. Clic derecho sobre el icono Mi Pc y seleccionar propiedades.
 2. Pestaña Opciones Avanzadas y pulsamos Variables de Entorno.
 3. En variables de usuario para Administrador añadimos dos nuevas variables:
 - GRIDGAIN_HOME: C:\Archivos de programa\gridgain-2.0.3
 - JAVA_HOME: C:\Archivos de programa\Java\jdk1.6.0en nuestro caso son las versiones 2.0.3 de GridGain y 1.6 de JDK.
- Mac OS X
 1. Abrir un terminal y crear carpeta .MacOSX en el directorio HOME. Los directorios serán en los que están instalados java y gridgain.

- `$mkdir ~/.MacOSX`

2. Crear el archivo `environment.plist`

- `$touch ~/.MacOSX/environment.plist`
- `$nano ~/.MacOSX/environment.plist` y pegar el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM "file:///localhost/System/Library/ \
DTDs/PropertyList.dtd">
<plist version="0.9">
<dict>
<key>JAVA\_HOME</key>
  <string>System/Library/Frameworks/JavaVM.framework\
/Versions/1.6.0/Home/</string>
  <key>GRIDGAIN\_HOME</key>
  <string>/Applications/gridgain-2.0.3/</string>
</dict>
</plist>
```

Entrar en modo root: `sudo su` y la contraseña, esto lo hacemos para que todos los usuarios lo puedan usar.

- `$touch .profile`
- `$nano .profile` y pegar el siguiente contenido:

```
export GRIDGAIN_HOME=/Applications/gridgain-2.0.3/
export JAVA_HOME=/System/Library/Frameworks/JavaVM.framework\
/Versions/1.6.0/Home/
export PATH=$PATH
```

3. Cerramos Sesión y al iniciar todo funcionará bien.

■ Linux

1. Abrir un terminal y añadir las variables del entorno a `/etc/environment`

- `JAVA_HOME=JAVA_HOME="/opt/jdk1.6.0"`
- `GRIDGAIN_HOME="/opt/gridgain-2.0.3"`

2. Guardamos y ya podremos usar Gridgain.

1.4. Ejemplo HolaMundo

1. Crear un proyecto Java llamado HolaMundo. Ir al menú *Archivo* -> *Nuevo Proyecto*, ver figura 1.

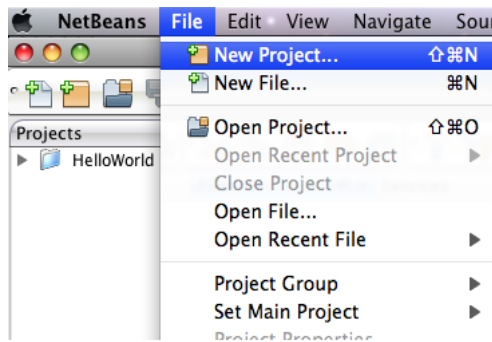


Figura 1: Nuevo Proyecto Java

2. Elegir el tipo del proyecto: *Java Application*, ver figura 2.

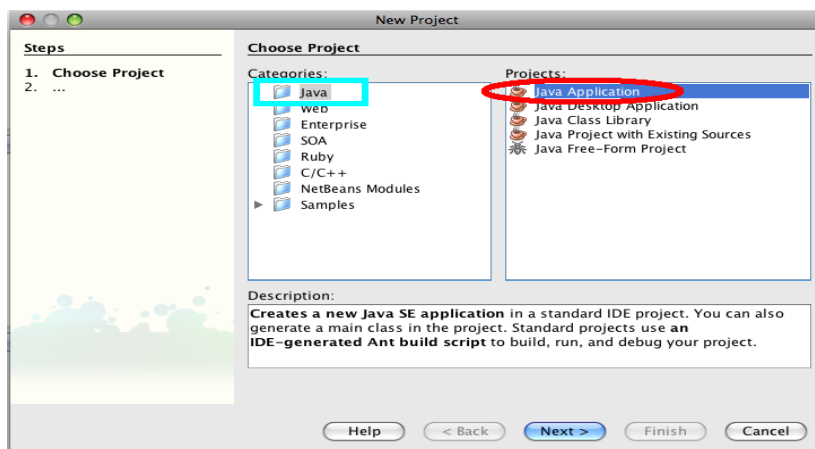


Figura 2: Java Application

3. Establecer el nombre del proyecto: *HolaMundo*, ver figura 3.

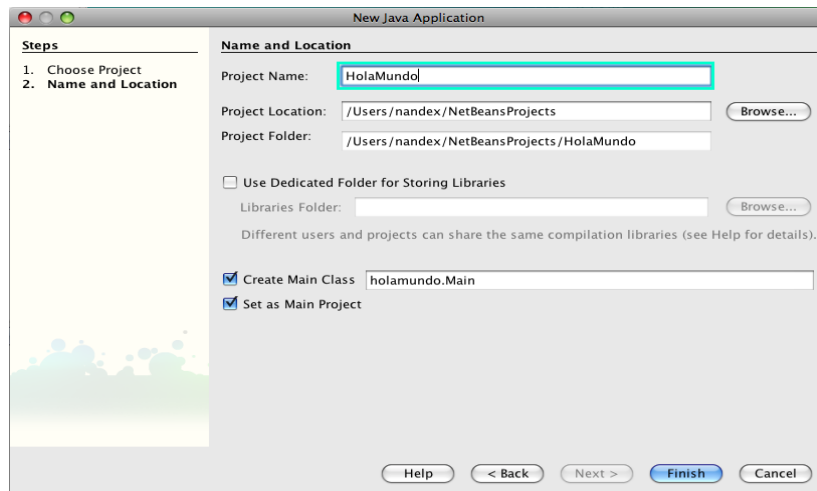


Figura 3: Crear proyecto HolaMundo

4. Crear un método al que llamamos *Say* que muestre por pantalla un parámetro de entrada de tipo String y en el método Main invocamos *Say*, de la forma de la figura 4.

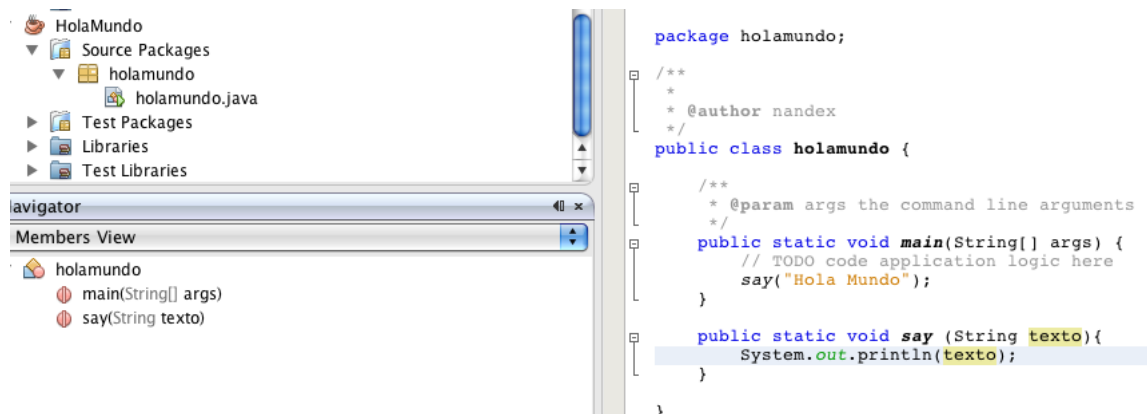


Figura 4: Método Say

5. Ejecutamos y comprobamos que todo funciona correctamente.
6. Ahora se lanzan los nodos Grid con Gridgain. Para ello vamos a la carpeta *bin* que esta dentro de GridGain y ejecutamos *gridgain.bat* (Windows) o *gridgain.sh* (GNU/Linux, Mac OS X).

7. Añadir todas las librerías del directorio *libs* de la carpeta de instalación de GridGain y el paquete *gridgain-2.0.3.jar* que se encuentra dentro de la misma, siguiendo las figuras 5 y 6. Añadimos lo siguiente:

```
public static void main (String[] args) throws GridException{
    GridFactory.start();
    try{
        say("Hola Mundo");}
    finally{
        GridFactory.stop(true);}
}
```

para lanzar la Grid sin olvidar las posibles excepciones. Además añadir los siguientes paquetes:

- `import org.gridgain.grid.GridException;`
- `import org.gridgain.grid.GridFactory;`

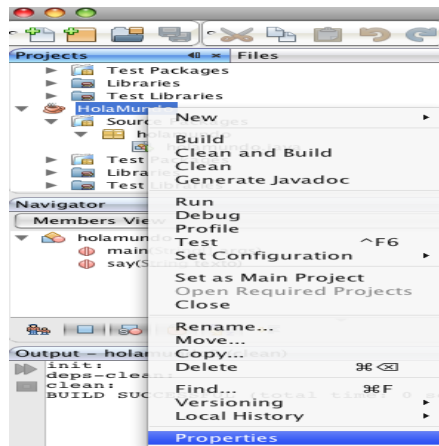


Figura 5: Menú Propiedades

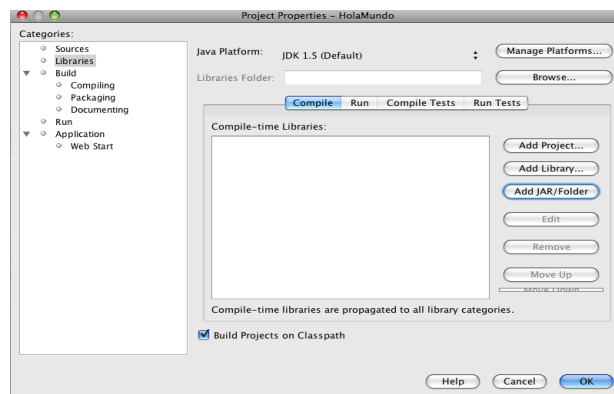


Figura 6: Add Jar/Folder

8. Comprobar que *Java Platform* en 'Libraries' y *Source/Binary format* en 'Sources' son el mismo que para Gridgain, siendo en este caso el contenido de la variable `JAVA_HOME`. Ejecutar y comprobar que la grid se ejecuta y lanza Hola Mundo. Podemos ver una topología de tres nodos con un nodo local y dos remotos, como aparece en la figura 7.

```

>> -----|
>> Topology Snapshot.
>> -----
>> Number of nodes: 3
>> Topology hash: 0xd59b9659
>> Local: 6b6a65fe-40d9-45df-a4e3-62d4eb75ac54, 192.168.1.4, Mac OS X i386
>> Remote: a4b03914-1df7-4997-a1bc-279411c38c77, 192.168.1.4, Mac OS X i386
>> Remote: 85bc3285-6d71-426f-b5eb-9dd22def0bae, 192.168.1.4, Mac OS X i386
>> Total number of CPUs: 2
>> Number of unique IP interfaces: 1
>> Unique IP interface: 192.168.1.4, en1, Mac OS X i386 10.5.4, 2 CPU(s)

```

Figura 7: Nodos Grid

1.5. Gridificando nuestra aplicación

Existen ahora mismo dos nodos corriendo de forma autónoma y hay que poner un tercer nodo en nuestra aplicación. Para gridificar con Gridgain tenemos usar dos técnicas:

1. Usar GridTask de forma Explícita. Para más información visitar:

<http://www.gridgain.com/javadoc/org/gridgain/grid/GridTask.html>.

2. Usar la notación `@Gridify` para indicar que queremos gridificar.

En esta ocasión optamos por la segunda opción (porque no necesitamos recibir nada de la ejecución), para lo que necesitamos añadir *VM Arguments* siguiendo las figuras 5 y 8. Añadiendo la dirección de gridgain de *aspectjweaver* de la forma:

-ea “-javaagent:/Applications/gridgain-2.0.3/libs/aspectjweaver-1.5.3.jar” and classpath aspectj. Con Gridify realizaremos una ejecución de lo que queramos en el método *execute* y cada una de las ejecuciones las trataremos en el método *reduce*. Si no se desea usar `@Gridify` se puede utilizar GridTask, para lo que se deber *Serializar* y obtener resultados de la Grid (usando los métodos de obtención, como *GridTaskFuture get()*).

En este caso no usaremos reduce puesto que lo que tenemos es una escritura en pantalla. Un ejemplo de GridTask usando Reduce es el siguiente:

- Calcular Matriz:

```
public void GridHistoNorm(Imagen img) throws GridException{
    //*****INICIO GRID*****//
    int minimo=0;
    //SI ES UNA IMAGEN VALIDA LANZA LA GRID.
    if((img.getNumFilas(>0) && (img.getNumColumnas(>0) {
        GridFactory.start();
    }
    try {
        Grid grid = GridFactory.getGrid();

        // EJECUTA EL METODO DE DIVISION DE LA IMAGEN
        GridTaskFuture<Integer> future = grid.execute(GridifyMesh.class, img);

        // OBTIENE EL RESULTADO.
        minimo= future.get();
    }
```

```

    }
    finally {
        GridFactory.stop(true);
    }
    //*****FIN GRID*****//
    ....
    ....
}

```

- Gridificar Matriz:

```

//RECIBE UNA MATRIZ Y UN ENTERO PARA DIVIDIRLA POR FILAS.
//Imagen es el parametro de entrada e Integer es lo que devuelve.
public class GridifyMesh extends GridTaskSplitAdapter<Imagen, Integer> {
    /** Grid logger. Log para las llamadas a metodos*/
        @GridLoggerResource
        private GridLogger log = null;
        int filas,columnas,i,j;
        int [][] datos;

        @Override
        public Collection<? extends GridJob> split(int gridSize, Imagen _img)
            throws GridException {
            filas=_img.getNumFilas();
            columnas=_img.getNumColumnas();
            datos=new int[filas][columnas];
            _img.grisesTOarray(datos);
            List<GridJob> jobs = new ArrayList<GridJob>(filas);

            for (i=0;i<filas;i++) {
//SE CREAN LOS TRABAJOS QUE SE VAN A LANZAR EN CADA NODO GRID.
                jobs.add(new GridJobAdapter<Integer>(i) {
//METODO EXECUTE QUE REALIZARA CADA NODO.
                    public Serializable execute() {
                        int i = getArgument();
                        int nummax=0;
//OBTENEMOS LA FILA Y REALIZAMOS CALCULOS PARA OBTENER UN NUMERO y SE DEVUELVE
                        return nummax;
                    }
                });
            }
        }
    }
}

```

```

    }
    return jobs;
}
//METODO PARA DEVOLVER EL VALOR DESEADO. SE OBTIENE EL VALOR DE EXECUTE
//CON ‘‘getData()’’ y DEVUELVE EL VALOR MINIMO.
@Override
public Integer reduce(List<GridJobResult> resultado) throws GridExcepti
    int min=0;
    // For the sake of this example, let’s sum all results.
    for (GridJobResult res : resultado) {
        if(min<(Integer)res.getData()){
            min=(Integer)res.getData();
        }
    }
    return min;
}
}

```

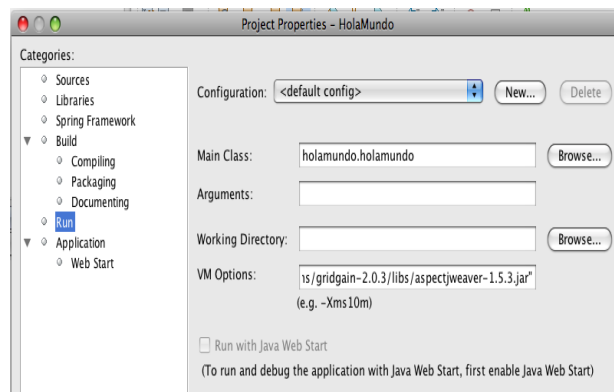


Figura 8: VM Argument

Además hay que incluir *aspectj* al Classpath. Para ello seguimos las figuras 5 y seleccionamos Libraries , Run y Add Jar/Folder siguiendo las figuras 9 y 10 e incluir */config/aop/aspectj*.

Ejecutamos y comprobamos que todavía se sigue usando un nodo para la ejecución. Seguimos con el ejemplo para Gridificar nuestra aplicación. Hasta ahora todas las ejecuciones se lanzan aleatoriamente sobre uno de los procesadores que tenemos en la estructura grid. Ha llegado el momento de añadir

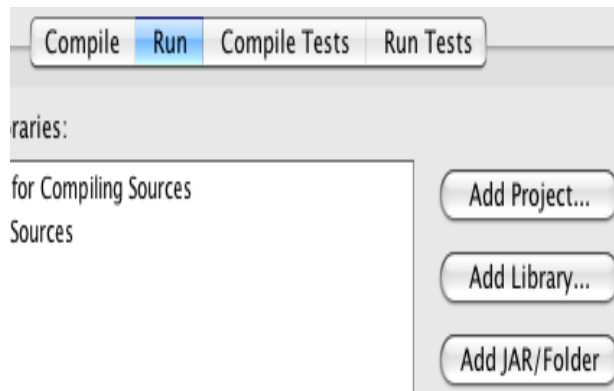


Figura 9: Menú Run

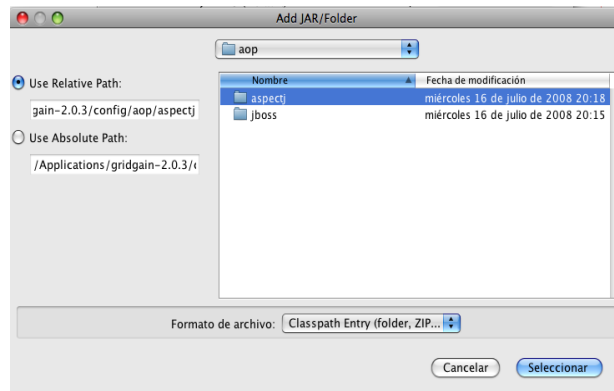


Figura 10: Add Jar/Folder al ClassPath

el gestor de tareas para dividir ejecución.

Creamos la clase *HolaMundoTask* que herede de *GridTaskSplitAdapter* y tenga la siguiente estructura:

1. HolaMundoTask.java:

```
package holamundo;
import java.util.*;
import org.gridgain.grid.*;
import org.gridgain.grid.GridJobAdapter;
import org.gridgain.grid.GridTaskSplitAdapter;
import org.gridgain.grid.gridify.*;
import java.io.Serializable;
public class HolaMundoTask extends GridTaskSplitAdapter<GridifyArgument, String>
** Log para la Grid **
@Override
//Siempre tendremos dos metodos: split para controlar \
las tareas que se quieren enviar a la estructura grid.
protected Collection<? extends GridJob> split(int gridSize, \
GridifyArgument arg1) throws GridException {
String[] words = ((String)arg1.getMethodParameters()[0]).split(" ");
List<GridJobAdapter <String>> jobs = new ArrayList <GridJobAdapter <String>>();
for (String word : words) {
    // Todos los jobs obtienen su palabra como un argumento
    jobs.add(new GridJobAdapter<String>(word){
        //Se ejecuta el metodo say con el argumento que se le pasa.
        public Serializable execute() throws GridException{
            // Ejecuta el metodo para gridificar.
            HolaMundo.say(getArgument());
            return null;
        }
    });
}
return jobs;
}
//Metodo Reduce por si queremos liberar la carga de trabajo.
public String reduce(List arg0) throws GridException {
    return null;
}
}
```

2. HolaMundo.java:

```
package holamundo;
```

```

import org.gridgain.grid.GridException;
import org.gridgain.grid.GridFactory;
import org.gridgain.grid.gridify.Gridify;
public class HolaMundo {
    public static void main (String[] args) throws GridException{
        GridFactory.start();
        try{
            say("Hola Mundo");
        }
        finally{
            GridFactory.stop(true);
        }
    }
    @Gridify(taskClass=HolaMundoTask.class)
    public static void say(String text){
        System.out.println("*** "+text+" ***");
    }
}

```

Ejecutamos y comprobamos que cada palabra se ejecuta en un nodo de la estructura. Si cambiamos el String por *Hola Mundo Feliz*, comprobamos que se muestra cada palabra en un nodo. Las palabras se van asignando a los nodos que tenemos según vayan terminando. En la figura 11 podemos ver la ejecución del proyecto con 3 palabras:

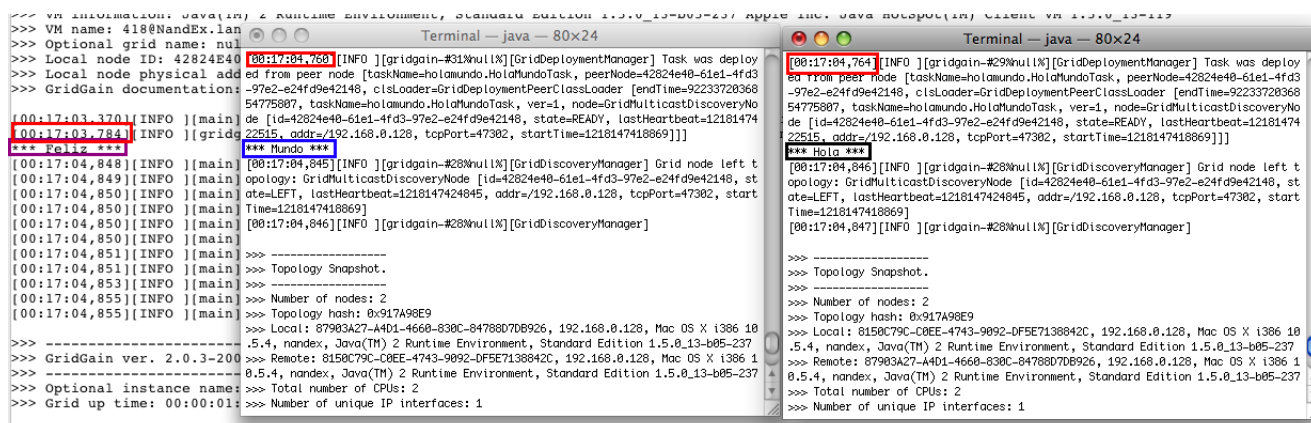


Figura 11: Ejecución Hola Mundo Feliz