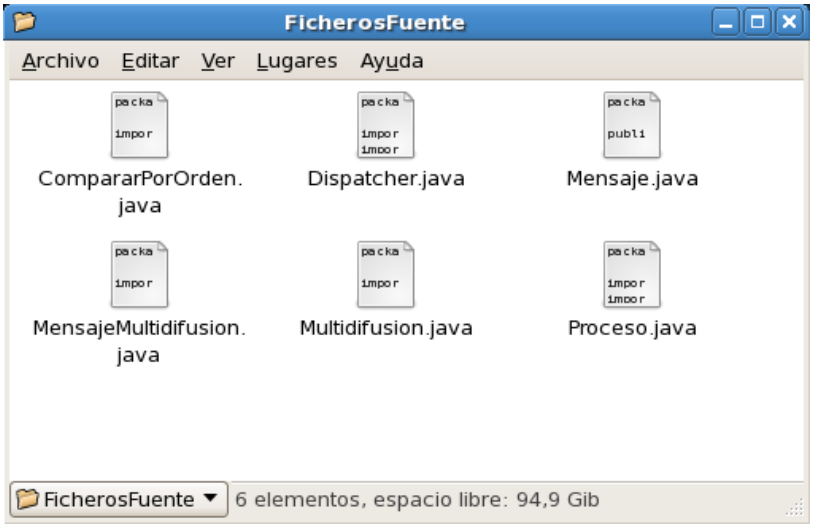
Ficheros Fuente



Los ficheros fuentes que marcan la funcionalidad del proyecto son:

Proceso.java

Esta clase extiende de Thread. Consta de un constructor en el que inicializa sus atributos según los argumentos que le pasa el Dispatcher. Tiene tres métodos que se corresponden con la funcionalidad del algoritmo Isis. Estos métodos son: mensaje, propuesta y acuerdo.

mensaje: en este método se comprueba si se desea realizar el algoritmo sin el protocolo de multidifusión ordenada o no. Dependiendo de si se realiza con el protocolo o no, el procedimiento es incrementar el orden (tiempo lógico), introducir el mensaje en la cola y enviar la propuesta al proceso que envió el mensaje o entregar el mensaje tan pronto como se recibe.

```
//RECEPCION MENSAJE
public void mensaje(String cadena_mensaje, String procesoEmisorId) {
	String idMensaje;
	Mensaje mensajeRecibido;

	if(ordenacionTotal.equalsIgnoreCase("false")) { //Se ejecuta SIN el protocolo de multidifusión ordenada
		String[] camposMensaje = cadena_mensaje.split(",");
		String contenido = camposMensaje[0];

		//Escritura en el fichero log del mensaje Pxx nnn
		pw.println(contenido);
		System.out.println(contenido);
		numMensajesEntregados++;

		//Comprobación de si se han recibido todos los mensajes del grupo para así cerrar el fichero
		if(numMensajesEntregados == (infoProcesos.length*100)) {
			pw.close();
			//Aviso al dispatcherArbitro
			int flag = 0;
			for(int j=0; j<infoProcesos.length && flag==0; j++) {
				if(infoProcesos[j][1].equals(dispatcherArbitro)) {
					System.out.println(service[j].path("rest/dispatcher/finalizacion").request(MediaType.TEXT_PLAIN).get(String.class));
					flag=1;
				}
			}
			System.out.println("Proceso "+infoProcesos[0][0]+" ha finalizado.");
		}

	}
```

```
                else if(ordenacionTotal.equalsIgnoreCase("true")) { //Se ejecuta CON el protocolo de multidifusión ordenada
                        synchronized(testigo) {
                                //Incremento del orden (LC1)
                                orden = orden + 1;

                                //Introducción del mensaje en la cola y reordenación de la cola
                                String[] camposMensaje = cadena_mensaje.split(",");
                                String contenido = camposMensaje[0];
                                idMensaje = camposMensaje[1];
                                String estado = camposMensaje[2];
                                String emisor = camposMensaje[3];
                                mensajeRecibido = new Mensaje(contenido,idMensaje,estado,emisor);
                                mensajeRecibido.orden = orden;

                                cola.add(mensajeRecibido);
                                Collections.sort(cola,comparador);


                        }//fin synchronized testigo

                        int flag=0;
                        //Envio de la propuesta al proceso que envió el mensaje
                        for(int j=0; j<infoProcesos.length && flag==0; j++) {
                                if(infoProcesos[j][0].equals(procesoEmisorId)) {
                                        MensajeMultidifusion hiloMensajeMultidifusion = new MensajeMultidifusion(mensajeRecibido.orden,idMensaje,infoProcesos[j][0],service
[j],this);

                                        hiloMensajeMultidifusion.start();
                                        flag=1;
                                }
                        }
                }
        }
```

propuesta: cuando se recibe una propuesta, primero se actualiza el orden y a continuación se elige el mayor orden para el mensaje indicado y se incrementan las propuestas recibidas para dicho mensaje. Por último, se comprueba si se han recibido todas las propuestas para ese mensaje y cuando así sea, se multidifunde el acuerdo.

```
 //RECEPCION PROPUESTA
 public void propuesta(String ordenPropuesto, String idMensaje) {
         int propuestasRecibidas = 0;
         float ordenAcordado = 0;

         synchronized(testigo) {
                        // Actualización del orden (LC2)
                        if(Float.parseFloat(ordenPropuesto) > orden) {
                                orden = (int) Float.parseFloat(ordenPropuesto) + Float.parseFloat(infoProcesos[0][0])/10 + 1;
                        }
                        else {
                                orden = orden + 1;
                        }

                        //Obtención del mensaje correspondiente al idMensaje
                        int flag = 0;
                        for(int i=0; i<cola.size() && flag==0; i++) {
                                if(cola.get(i).id_mensaje.equals(idMensaje)) {
                                        //Elección del mayor orden
                                        cola.get(i).orden = Math.max(cola.get(i).orden, Float.parseFloat(ordenPropuesto));
                                        ordenAcordado = cola.get(i).orden;

                                        //Incremento de las propuestas recibidas
                                        cola.get(i).numPropuestasRecibidas++;
                                        propuestasRecibidas = cola.get(i).numPropuestasRecibidas;

                                        flag=1;
                                }
                        }
         } //fin synchronized testigo

         //Comprobación de si se han recibido todas las propuestas
         if(propuestasRecibidas == infoProcesos.length) {
                 Semaphore semAviso = new Semaphore(0,true);
                 MensajeMultidifusion[] hilosMensajeMultidifusion = new MensajeMultidifusion[infoProcesos.length];

                 //Multidifusión del acuerdo
                 for(int j=0; j<infoProcesos.length; j++) {
                         hilosMensajeMultidifusion[j] = new MensajeMultidifusion(ordenAcordado,idMensaje,infoProcesos[j][0],service[j],this,semAviso);
                         hilosMensajeMultidifusion[j].start();
                 }

                 semAviso.release(infoProcesos.length);
         }

 }
```

acuerdo: si un proceso recibe un acuerdo, actualiza su tiempo lógico y marca como definitivo el mensaje para el cual ha recibido el acuerdo. Finalmente, comprueba si puede entregar mensajes de la cola. Si ha realizado la entrega de todos los mensajes, avisa a uno de los dispatcher (que será el dispatcher Arbitro) para así indicarle que ha finalizado.

```
//RECEPCION ACUERDO
public void acuerdo(String ordenAcordado, String idMensaje) {

	synchronized(testigo) {
			// Actualización del orden (LC2)
			if(Float.parseFloat(ordenAcordado) > orden) {
				orden = (int) Float.parseFloat(ordenAcordado) + Float.parseFloat(infoProcesos[0][0])/10 + 1;
			}
			else {
				orden = orden + 1;
			}

			//Obtención del mensaje correspondiente al idMensaje
			int flag=0;
			for(int i=0; i<cola.size() && flag==0; i++) {
				if(cola.get(i).id_mensaje.equals(idMensaje)) {
					cola.get(i).orden = Float.parseFloat(ordenAcordado);
					cola.get(i).estado = "DEFINITIVO";
					flag=1;
				}
			}

			//Reordenación de la cola
			Collections.sort(cola,comparador);

			flag = 0;
			//Si hay mensajes en estado DEFINITIVO en la cola, se realiza la Entrega
			while(!cola.isEmpty() && flag==0) {
				if(cola.get(0).estado.equals("DEFINITIVO")) {
					//Escritura en el fichero log del mensaje Pxx nnn
					pw.println(cola.get(0).contenido+" "+cola.get(0).orden);
					System.out.println(cola.get(0).contenido+" "+cola.get(0).orden);
					numMensajesEntregados++;
					//Eliminación del mensaje entregado de la cola
				cola.remove(0);
				}
				else {
					flag=1;
				}
			}

			//Comprobación de si se han recibido todos los mensajes del grupo para así cerrar el fichero
			if(numMensajesEntregados == (infoProcesos.length*100)) {
				pw.close();
				//Aviso al dispatcherArbitro
				flag = 0;
				for(int j=0; j<infoProcesos.length && flag==0; j++) {
					if(infoProcesos[j][1].equals(dispatcherArbitro)) {
						System.out.println(service[j].path("rest/dispatcher/finalizacion").request(MediaType.TEXT_PLAIN).get(String.class));
						flag=1;
					}
				}
				System.out.println("Proceso "+infoProcesos[0][0]+" ha finalizado.");
			}
```

En el método run del proceso, éste realiza una llamada al servicio sincronización del dispatcher Arbitro para que de esta manera todos los procesos comiencen a multidifundir a la vez.

```
//METODO RUN
public void run() {

	// Llamada al servicio de sincronizacion del dispatcher arbitro
	// para que cuando ya todos esten listos comiencen a multidifundir
	int flag = 0;
	for(int j=0; j<infoProcesos.length && flag==0; j++) {
		if(infoProcesos[j][1].equals(dispatcherArbitro)) {
			System.out.println(service[j].path("rest/dispatcher/sincronizacion").request(MediaType.TEXT_PLAIN).get(String.class));
			hiloMultidifusion.start();
			flag=1;
		}
	}
}
```

Multidifusion.java

La clase multidifusión también extenderá de Thread. Cada proceso crea un hiloMultidifusion que será realmente el encargado de realizar la multidifusión de los 100 mensajes a los 6 procesos.

```
//METODO RUN
public void run() {

        String contenido, id_mensaje, cadenaMensaje, emisor;
        Semaphore semAviso = new Semaphore(0,true);

        //Envío de 100 mensajes al grupo multicast
        for(int i=0; i<100; i++) {

                semAviso = new Semaphore(0,true);

                //Construcción del mensaje
                contenido = "P"+p.infoProcesos[0][0]+" "+i;
                id_mensaje = i+"-"+p.infoProcesos[0][0];
                emisor = p.infoProcesos[0][0];

                cadenaMensaje = contenido+","+id_mensaje+","+"PROVISIONAL"+","+emisor;

                //Multidifusión del mensaje
                MensajeMultidifusion[] hilosMensajeMultidifusion = new MensajeMultidifusion[p.infoProcesos.length];
                for(int j=0; j<p.infoProcesos.length; j++) {
                        hilosMensajeMultidifusion[j] = new MensajeMultidifusion(cadenaMensaje,p.infoProcesos[j][0],service[j], p, semAviso);
                        hilosMensajeMultidifusion[j].start();
                }

                semAviso.release(p.infoProcesos.length);

                //Duerme un tiempo aleatorio entre 1 y 1.5 s
                try { Thread.sleep(1000+(int)(Math.random()*500)); } catch (InterruptedException e) {e.printStackTrace();}

        }

        System.out.println("Hilo multidifusion del proceso "+p.infoProcesos[0][0]+" ha finalizado.");

}
```

Por cada mensaje, crea 6 hilos, uno para cada envío a cada proceso; este hilo es el que realiza el envío real del mensaje (ya sea un mensaje, una propuesta o un acuerdo). Si el envío es al propio proceso, éste se realiza de manera local.

```
        //METODO RUN
        public void run() {

                //Envío de un mensaje
                if(tipoMensaje == 1) {
                        //Espero
                        try {semAviso.acquire();} catch (InterruptedException e1) {e1.printStackTrace();}

                        if(p.infoProcesos[0][0].equals(procesoDestino)) { //Envío a nosotros mismos
                                p.mensaje(cadenaMensaje, procesoDestino);
                        }
                        else {
                                //Duerme un tiempo aleatorio entre 0.2 y 0.5 s
                                try { Thread.sleep(200+(int)(Math.random()*300)); } catch (InterruptedException e) {e.printStackTrace();}

                                System.out.println(service.path("rest/dispatcher/mensaje").queryParam("cadenaMensaje", cadenaMensaje).queryParam("procesoDestino",
procesoDestino).request(MediaType.TEXT_PLAIN).get(String.class));
                        }
                }

                //Envío de una propuesta
                else if(tipoMensaje == 2) {
                        if(p.infoProcesos[0][0].equals(procesoDestino)) { //Envío a nosotros mismos
                                p.propuesta(""+orden, idMensaje);
                        }
                        else {
                                //Duerme un tiempo aleatorio entre 0.2 y 0.5 s
                                try { Thread.sleep(200+(int)(Math.random()*300)); } catch (InterruptedException e) {e.printStackTrace();}

                                System.out.println(service.path("rest/dispatcher/propuesta").queryParam("ordenPropuesto", ""+orden).queryParam("idMensaje", idMensaje).queryParam
("procesoDestino", procesoDestino).request(MediaType.TEXT_PLAIN).get(String.class));
                        }
                }

                //Envío de un acuerdo
                else if(tipoMensaje == 3) {
                        //Espero
                        try {semAviso.acquire();} catch (InterruptedException e1) {e1.printStackTrace();}

                        if(p.infoProcesos[0][0].equals(procesoDestino)) { //Envío a nosotros mismos
                                p.acuerdo(""+orden, idMensaje);
                        }
                        else {
                                //Duerme un tiempo aleatorio entre 0.2 y 0.5 s
                                try { Thread.sleep(200+(int)(Math.random()*300)); } catch (InterruptedException e) {e.printStackTrace();}

                                System.out.println(service.path("rest/dispatcher/acuerdo").queryParam("ordenAcordado", ""+orden).queryParam("idMensaje", idMensaje).queryParam
("procesoDestino", procesoDestino).request(MediaType.TEXT_PLAIN).get(String.class));
                        }
                }
        }
}
```

Dispatcher.java

El dispatcher tiene distintos servicios a los cuales se puede acceder:

iniciarProcesos: a cada dispatcher de cada una de las 3 máquinas se le pasan los argumentos necesarios para que cada uno cree 2 procesos que se encontrarán en la misma máquina que ellos.

```
@Path("iniciarProcesos")
@GET
@Produces(MediaType.TEXT_PLAIN)
public String iniciarProcesos(@QueryParam(value="dispatcherArbitro") String dispatcherArbitro,
                @QueryParam(value="id1") String id1,
                @QueryParam(value="ip1") String ip1,
                @QueryParam(value="id2") String id2,
                @QueryParam(value="ip2") String ip2,
                @QueryParam(value="id3") String id3,
                @QueryParam(value="ip3") String ip3,
                @QueryParam(value="id4") String id4,
                @QueryParam(value="ip4") String ip4,
                @QueryParam(value="id5") String id5,
                @QueryParam(value="ip5") String ip5,
                @QueryParam(value="id6") String id6,
                @QueryParam(value="ip6") String ip6,
                @QueryParam(value="rutaFichero") String rutaFichero,
                @QueryParam(value="ordenacionTotal") String ordenacionTotal) {

        Proceso p1 = new Proceso(dispatcherArbitro,id1,ip1,id2,ip2,id3,ip3,id4,ip4,id5,ip5,id6,ip6,rutaFichero,ordenacionTotal);
        Proceso p2 = new Proceso(dispatcherArbitro,id2,ip2,id1,ip1,id3,ip3,id4,ip4,id5,ip5,id6,ip6,rutaFichero,ordenacionTotal);

        procesos.put(id1, p1);
        procesos.put(id2, p2);
        procesos.get(id1).start();
        procesos.get(id2).start();

        return "Procesos iniciados.";

}
```

sincronización: a uno de los dispatchers (será considerado por todos como el dispatcherArbitro), cada proceso le indica que desea sincronizarse con el resto de procesos de tal manera que todos se quedan esperando en un semáforo hasta que el último avisa al resto y de esta manera todos comienzan a multidifundir.

```
@Path("sincronizacion")
@GET
@Produces(MediaType.TEXT_PLAIN)
public String sincronizacion() {

        try { semAtomicoPreparados.acquire();} catch (InterruptedException e) {e.printStackTrace();}

        numPreparados++;

        if(numPreparados == 6) {
                semAtomicoPreparados.release();
                //Como soy el último, aviso al resto
                semPreparados.release(5);
        }
        else {
                semAtomicoPreparados.release();
                //Espero al resto de procesos
                try { semPreparados.acquire();} catch (InterruptedException e) {e.printStackTrace();}

        }

        return "Sincronizacion realizada.";
}
```

mensaje, propuesta, acuerdo: estos servicios son realmente servicios envoltorio mediante los cuales cada dispatcher redirige el mensaje, la propuesta o el acuerdo a uno de los dos procesos que se encuentran en la misma máquina.

```
@Path("mensaje")
@GET
@Produces(MediaType.TEXT_PLAIN)
public String mensaje(@QueryParam(value="cadenaMensaje") String cadenaMensaje,
                @QueryParam(value="procesoDestino") String procesoDestino) {

        // Redirige el mensaje al procesoDestino
        String[] camposMensaje = cadenaMensaje.split(",");
        String procesoEmisorId = camposMensaje[3];
        procesos.get(procesoDestino).mensaje(cadenaMensaje, procesoEmisorId);

        return "Mensaje "+camposMensaje[1]+" redirigido al proceso "+procesoDestino;

}

@Path("propuesta")
@GET
@Produces(MediaType.TEXT_PLAIN)
public String propuesta(@QueryParam(value="ordenPropuesto") String ordenPropuesto,
                @QueryParam(value="idMensaje") String idMensaje,
                @QueryParam(value="procesoDestino") String procesoDestino) {

        // Redirige la propuesta al procesoDestino
        procesos.get(procesoDestino).propuesta(ordenPropuesto, idMensaje);

        return "Propuesta redirigida al proceso "+procesoDestino+" idMensaje: "+idMensaje+" ordenPropuesto: "+ordenPropuesto;

}

@Path("acuerdo")
@GET
@Produces(MediaType.TEXT_PLAIN)
public String acuerdo(@QueryParam(value="ordenAcordado") String ordenAcordado,
                @QueryParam(value="idMensaje") String idMensaje,
                @QueryParam(value="procesoDestino") String procesoDestino) {

        // Redirige el acuerdo al procesoDestino
        procesos.get(procesoDestino).acuerdo(ordenAcordado, idMensaje);

        return "Acuerdo redirigido al proceso "+procesoDestino+" idMensaje: "+idMensaje+" ordenAcordado: "+ordenAcordado;

}
```

finalización, fin: al servicio finalización del dispatcherArbitro acceden los procesos para indicar que han realizado la entrega de todos los mensajes y esto se simboliza haciendo un release en el semáforo de finalización. Al servicio fin del dispatcherArbitro se accede desde el script de lanzamiento y así de esta manera sabremos cuando el algoritmo isis ha finalizado y esto ocurrirá cuando todos los procesos hayan hecho un release en el semáforo de finalización.

```
@Path("finalizacion")
@GET
@Produces(MediaType.TEXT_PLAIN)
public String finalizacion() {
        //Un proceso avisa cuando ha finalizado
        semFinalizacion.release();

        return "";
}

@Path("fin")
@GET
@Produces(MediaType.TEXT_PLAIN)
public String fin() {
        //Espera a que los 6 procesos hayan finalizado
        try { semFinalizacion.acquire(6);} catch (InterruptedException e) {e.printStackTrace();}

        return "El algoritmo ISIS ha finalizado.";
}
```

Scripts

En el script de lanzamiento, se realiza el despliegue en las 3 máquinas, se arranca tomcat en cada una de ellas, se inicializa a los procesos en cada dispatcher y por tanto, comienza el

algoritmo isis. Por último, se accede al servicio fin del dispatcherArbitro para esperar a que el algoritmo haya finalizado.

Uso: `./lanzamiento.sh <ipServidor1> <ipServidor2> <ipServidor3> <rutaFichero> <true/false> <rutaTomcat>`

```
        #1) Despliegue de los archivos .war a la ruta de tomcat
        chmod +x ./algoritmoISIS.war
        echo ""
        echo "Realizando el despliegue..."
        echo ""
                #1.1) Limpiamos el codigo anterior
                ssh $1 "rm $6/apache-tomcat-7.0.61/webapps/algoritmoISIS.war; exit"
                ssh $2 "rm $6/apache-tomcat-7.0.61/webapps/algoritmoISIS.war; exit"
                ssh $3 "rm $6/apache-tomcat-7.0.61/webapps/algoritmoISIS.war; exit"
                #1.2) Mandamos el nuevo codigo
                scp ./algoritmoISIS.war $1:$6/apache-tomcat-7.0.61/webapps/
                scp ./algoritmoISIS.war $2:$6/apache-tomcat-7.0.61/webapps/
                scp ./algoritmoISIS.war $3:$6/apache-tomcat-7.0.61/webapps/

        #2) Arranque de tomcat en las maquinas
        echo ""
        echo "Arrancando tomcat..."
        echo ""
        ssh $1 "export PATH=$PATH:/opt/jdk1.6.0/bin; chmod +x $6/apache-tomcat-7.0.61/bin/startup.sh; chmod +x $6/apache-tomcat-7.0.61/bin/catalina.sh; $6/apache-tomcat-7.0.61/
bin/startup.sh; exit"
        ssh $2 "export PATH=$PATH:/opt/jdk1.6.0/bin; chmod +x $6/apache-tomcat-7.0.61/bin/startup.sh; chmod +x $6/apache-tomcat-7.0.61/bin/catalina.sh; $6/apache-tomcat-7.0.61/
bin/startup.sh; exit"
        ssh $3 "export PATH=$PATH:/opt/jdk1.6.0/bin; chmod +x $6/apache-tomcat-7.0.61/bin/startup.sh; chmod +x $6/apache-tomcat-7.0.61/bin/catalina.sh; $6/apache-tomcat-7.0.61/
bin/startup.sh; exit"

        sleep 15

        #3) Lanzamos los procesos, y por tanto, comienza el algoritmo ISIS
        echo ""
        echo "Iniciando algoritmo..."
        echo ""
        inicio1="http://$1:8080/algoritmoISIS/rest/dispatcher/iniciarProcesos?dispatcherArbitro=$1&id1=1&ip1=$1&id2=2&ip2=$1&id3=3&ip3=$2&id4=4&ip4=$2&id5=5&ip5=$3&id6=6&ip6=
$3&rutaFichero=$4/&ordenacionTotal=$5"
        curl $inicio1
        echo ""
        inicio2="http://$2:8080/algoritmoISIS/rest/dispatcher/iniciarProcesos?dispatcherArbitro=$1&id1=3&ip1=$2&id2=4&ip2=$2&id3=1&ip3=$1&id4=2&ip4=$1&id5=5&ip5=$3&id6=6&ip6=
$3&rutaFichero=$4/&ordenacionTotal=$5"
        curl $inicio2
        echo ""
        inicio3="http://$3:8080/algoritmoISIS/rest/dispatcher/iniciarProcesos?dispatcherArbitro=$1&id1=5&ip1=$3&id2=6&ip2=$3&id3=1&ip3=$1&id4=2&ip4=$1&id5=3&ip5=$2&id6=4&ip6=
$2&rutaFichero=$4/&ordenacionTotal=$5"
        curl $inicio3

        echo ""
        echo ""
        echo "Esperando..."
        echo ""
        fin="http://$1:8080/algoritmoISIS/rest/dispatcher/fin"
        curl $fin
        echo ""

else
```

En el script de comprobación, se finaliza tomcat en cada una de las máquinas, se realiza la recogida de los logs de cada máquina y se comprueba si el contenido de estos logs es igual o no.

Uso: `./comrpobacion.sh <ipServidor1> <ipServidor2> <ipServidor3> <rutaFichero> <nombreCarpetaFicherosRecogidos> <rutaTomcat>`

```
        #1) Finalización de tomcat en las máquinas
        echo ""
        echo "Finalizando tomcat..."
        echo ""
        ssh $1 "export PATH=$PATH:/opt/jdk1.6.0/bin; chmod +x $6/apache-tomcat-7.0.61/bin/shutdown.sh; chmod +x $6/apache-tomcat-7.0.61/bin/catalina.sh; $6/apache-tomcat-7.0.61/
bin/shutdown.sh; exit"
        ssh $2 "export PATH=$PATH:/opt/jdk1.6.0/bin; chmod +x $6/apache-tomcat-7.0.61/bin/shutdown.sh; chmod +x $6/apache-tomcat-7.0.61/bin/catalina.sh; $6/apache-tomcat-7.0.61/
bin/shutdown.sh; exit"
        ssh $3 "export PATH=$PATH:/opt/jdk1.6.0/bin; chmod +x $6/apache-tomcat-7.0.61/bin/shutdown.sh; chmod +x $6/apache-tomcat-7.0.61/bin/catalina.sh; $6/apache-tomcat-7.0.61/
bin/shutdown.sh; exit"

        #2) Recogida de logs
        echo ""
        echo "Recogiendo logs..."
        echo ""
        mkdir ./$5
        scp $1:$4/*.txt ./$5
        scp $2:$4/*.txt ./$5
        scp $3:$4/*.txt ./$5

        #3) Comprobacion de ficheros
        echo ""
        echo "Comprobando logs..."
        echo ""
        diff -q ./$5/fichero1.txt ./$5/fichero2.txt
        diff -q ./$5/fichero1.txt ./$5/fichero3.txt
        diff -q ./$5/fichero1.txt ./$5/fichero4.txt
        diff -q ./$5/fichero1.txt ./$5/fichero5.txt
        diff -q ./$5/fichero1.txt ./$5/fichero6.txt
```