

Alumnos:

Juan Marcelo Luvían Mendoza
Jesus Alberto Rodríguez Hernández
Jorge Luis Jácome Domínguez
Eduardo Loyo Martínez
Julián Galván Viveros

1.0.0. Introducción

Este documento del marco de trabajo de procesamiento en paralelo de Java (Java Parallel Processing Framework por sus siglas en inglés JPPF) explica el funcionamiento de JPPF mediante un ejemplo elaborado en Java, el cual se pondrá en práctica conforme a lo investigado en la documentación de JPPF, el documento se conforma de los siguientes componentes en este marco de trabajo:

- Arquitectura.
- Funcionamiento.
- Casos de estudio.
- Evaluación.
- Conclusiones de la investigación de JPPF.

JPPF permite, a las aplicaciones con grandes necesidades de potencia y procesamiento que van a ser ejecutadas en cualquier número de computadoras, reducir dramáticamente su tiempo de procesamiento, esto se hace mediante el fraccionamiento de una aplicación en partes más pequeñas que puedan ser ejecutadas simultáneamente en distintas máquinas.

En el funcionamiento de JPPF se toman en cuenta 2 aspectos:

- La división de de una aplicación en partes más pequeñas que se pueden ejecutar de manera independiente y en paralelo, esto da como resultado un objeto JPPF conocido como “Job” la cual a su vez contiene partes independientes llamadas “tasks”.
- La ejecución de la aplicación en el Grid de JPPF es muy simple, puesto que permite conectar equipos y compartir recursos no centralizados gráficamente, esto es, redes de gran magnitud, permitiendo que cierta cantidad de recursos que el administrador del esclavo designe para compartirlos a la red y los demás recursos servirán para que los equipos mantengan su funcionalidad normal.

Para comprender de una mejor forma el funcionamiento de JPPF el equipo puso en ejecución los archivos descargados desde la página www.jppf.org, todos estos

archivos ejecutados en sistemas linux, utilizando 5 equipos portátiles conectados a la misma red para lograr obtener resultados que serán explicados más adelante.

En el ejemplo proporcionado se mostrarán más detalles sobre el uso de JPPF, además se explicarán ventajas y desventajas sobre el mismo.

2.0.0. Arquitectura de JPPF (Java Parallel Processing Framework).

Como ya se mencionó en la introducción, los frameworks de desarrollo sobre tecnologías GRID tienen dos características importantes:

1. La capacidad de dividir una aplicación en partes más pequeñas que se puedan ejecutar de forma autónoma y paralela.
2. Crear un entorno para ejecutar estas aplicaciones y proporcionar herramientas de gestión.

En la primera característica se ofrece una abstracción de los problemas denominada Jobs (Trabajos) que está compuesta a su vez de Tasks (Tareas) mismas que pueden ejecutarse de forma independiente.

Para la segunda característica, tenemos que JPPF proporciona un entorno de ejecución con los siguientes elementos que integran su arquitectura:

- Nodo
- Servidor
- Herramienta de monitorización

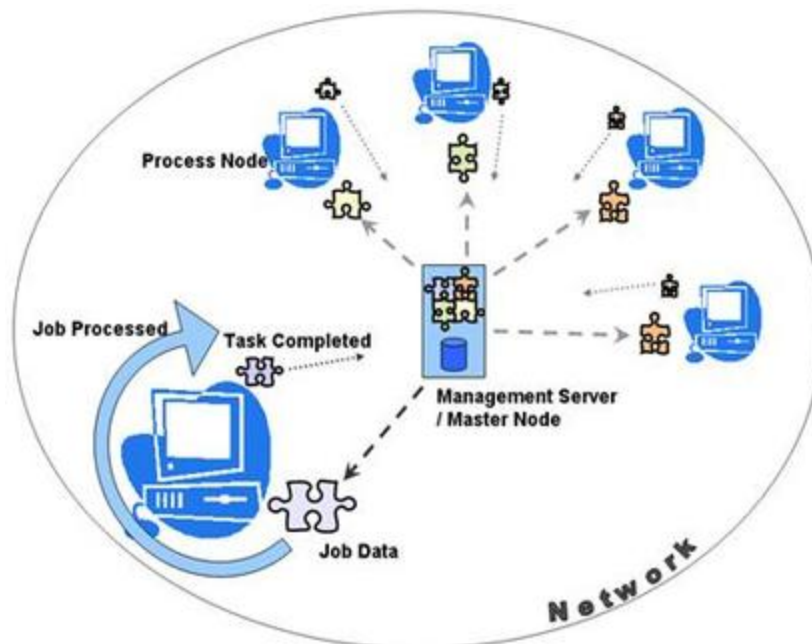
Esta arquitectura permite la distribución de los diferentes trabajos que residen en los servidores a los nodos que se ejecutan en máquinas diferentes e independientes.

Es recomendable (para tener en funcionamiento una instalación mínima) ejecutar un servidor y un nodo en una misma máquina, o bien dos máquinas conectadas en la misma subred TCP, el servidor se convertirá en ese momento en el punto de acceso de todos los trabajos que se deseen ejecutar.

El modelo de comunicación utilizado por JPPF es RMI (Por sus siglas en inglés Remote Method Invocation, Invocación de Métodos Remotos). RMI es el método más sencillo para la comunicación entre programas de Java, debido a que está integrado en la propia tecnología de Java. Tanto los nodos como los servidores disponen de su propio registro RMI (RMIRegistry) que permite informar de los servicios disponibles. Los puertos utilizados para comunicación entre los diferentes componentes se definen en el fichero de configuración del elemento de monitorización.

El framework viene distribuido en módulos diferentes. Cada uno de estos módulos tiene un objetivo y puede ser ejecutado de manera individual, dependiendo de las necesidades:

1. Driver: Es el nombre que recibe el servidor de la infraestructura, por esta razón también se le conoce como servidor.
2. Nodo: Es el paquete que contiene los ejecutables para la infraestructura de los nodos.
3. Consola: Es el nombre que recibe la interfaz gráfica de usuario (GUI) para la monitorización y administración del framework.
4. Herramienta de monitorización: permite soportar las abstracciones de Jobs y Tasks.



Arquitectura JPPF. Imágen tomada de <https://internetenunclic.wikispaces.com/grid>

DESVENTAJAS

1. Recursos heterogéneos: JPPF debe ser capaz de poder utilizar cualquier tipo de recurso que maneje el sistema, si no se podrá hacer nada.
2. Hay varios **recursos** que deben controlarse externamente y que influyen en el funcionamiento del grid.
3. Necesidad de **desarrollo de aplicaciones** para usar el grid.
4. Comunicación lenta y no uniforme.

VENTAJAS

1. En definitiva, grid supone un avance respecto a la World Wide Web:
2. El World Wide Web proporciona un acceso a información que está almacenada en millones de ordenadores repartidos por todo el mundo.
3. El grid es una infraestructura nueva que proporciona acceso a potencia de cálculo y capacidad de almacenamiento distribuida por una organización o por todo el mundo.

2.1.0. API

El framework de carga distribuida de clases en **JPPF** es el mecanismo que hace posible ejecutar el código en un nodo que no ha sido desarrollado específicamente para ese ambiente. A través de tasks, cuyo código solo esta definido en la aplicación cliente de JPPF, puede ser ejecutada en nodos remotos sin que el desarrollador se tenga que preocupar acerca de cómo se transporta el código.

mientras esté mecanismos es completamente transparente desde la perspectiva de la aplicación cliente, realmente tiene un número de implicaciones y particularidades que podrían impactar en varios aspectos de la ejecución del **JPPF tasks**, incluyendo rendimiento e integración con librerías externas.

abajo se muestra un ejemplo pequeño del camino que sigue una clase al ser cargada al mismo tiempo que una JPPF task es ejecutada en un nodo:

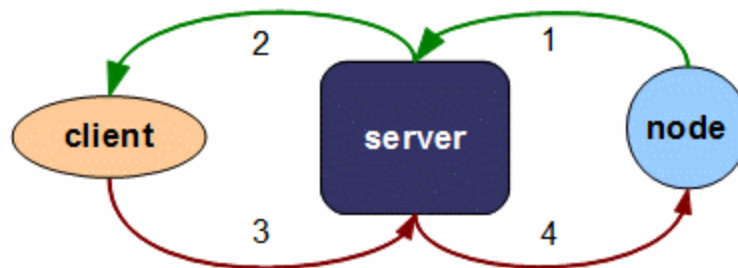


figura 2.1.0 diagrama clase

Podemos ver que el cargado de esta clase se hace en 4 pasos:

1. El nodo envía una petición de red al servidor remoto por la clase.
2. El servidor reenvía la petición hacia el cliente remoto identificado.
3. El cliente proporciona una respuesta (el bytecode de la clase) hacia el servidor.
4. El servidor reenvía la respuesta al hacia el nodo.

una vez que estos pasos son realizados, el nodo mantiene el bytecode de la clase y puede eficientemente definir y cargar, tal como lo haría con cualquier otra clase JAVA estándar.

2.2.0. Jerarquía de la clase Cargador en los nodos JPPF

El mecanismo de la clase cargador sigue una jerarquía basada en una relación padre-hijo entre instancias de la clase loader(cargador), como se mostrará en la *figura 2.2.0*:

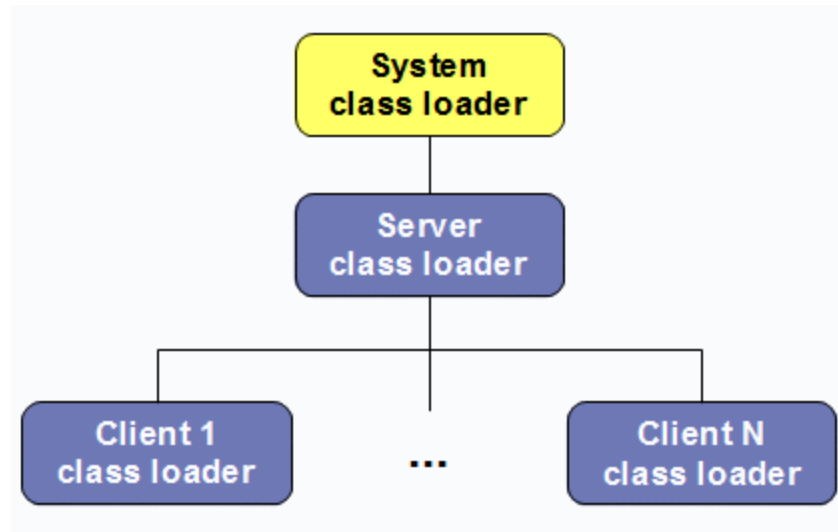
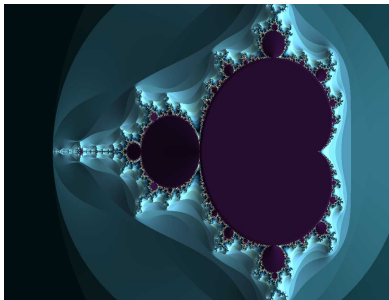
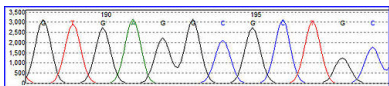



figura 2.2.0 Diagrama jerarquía clase loader(cargador)

2.3.0. Ejemplos Aplicaciones de JPPF

Aplicación	Actividad	Ejemplo
Fractales de Mandelbrot	Generación	
ADN	Secuenciación y Alineamiento de Proteínas	

Motor Web y Rastreador web	Motor de búsqueda web	
----------------------------	-----------------------	---

3.0.0. Ventajas y desventajas.

3.1.0. Ventajas

Entre las ventajas que se tienen de utilizar JPPF, tenemos las siguientes:

- Facilidad de instalación.
- Implementación de componentes JPPF más de un grupo es tan simple como copiar archivos a través de FTP o cualquier otro sistema de archivos de red.
- Permite a los desarrolladores centrarse en su núcleo de desarrollo de software, en lugar de perder el tiempo en la complejidad de procesamiento paralelo y distribuido.

3.2.0. Desventajas

- Recursos heterogéneos: JPPF debe ser capaz de poder utilizar cualquier tipo de recurso que maneje el sistema, si no se podrá hacer nada.
- Hay varios **recursos** que deben controlarse externamente y que influyen en el funcionamiento del grid.
- Necesidad de **desarrollo de aplicaciones** para usar el grid.
- Comunicación lenta y no uniforme.

4.0.0. Casos de estudio.

Entre algunas de las más populares implementaciones de JPPF esta su uso en servidores que utilicen la tecnología Java. Por ejemplo en servidores WEB que trabajan con Java como “.jsf” o “.jsp”, dado que el procesamiento de las peticiones del usuario pueden ser procesadas por distintos nodos y de acuerdo a la configuración de JPPF las respuestas pueden ser emitidas por distintos nodos. Esta misma aplicación de la distribución del trabajo en un servidor web es un tema estudiado en JPPF, de hecho

entre la documentación de JPPF existen ejemplo sobre como es aplicable JPPF en servidores WEB y servidores FTP.

5.0.0. Implementación de una aplicación.

La siguiente representación corresponde al proyecto de ejemplo de multiplicación de matriz incluido con JPPF. El fin de esta aplicación ejemplo es procesar la carga de trabajo de multiplicar la matriz en los distintos nodos disponibles en el Grid de computadoras JPPF.

El algoritmo empleado en el proyecto de ejemplo de multiplicación de matriz de JPPF trabaja con una matriz con tamaño modificable mediante la edición del código fuente, así como trabaja con un número de distribuciones de trabajo editable por código fuente. Nótese que el proyecto original por defecto no emite ningún mensaje sobre el nodo en el cual esta trabajando, el proyecto se limita devolver en la máquina de origen los resultados de las iteraciones realizadas.

El proyecto “Dense Matrix Multiplication sample” puede ser descargado desde la siguiente liga:

<http://sourceforge.net/projects/jppf-project/files/jppf-project/jppf%205.0/JPPF-5.0-samples-pack.zip/download>

y la documentación necesaria para entenderlo en la siguiente liga:

<http://www.jppf.org/samples-pack/MatrixMultiplication/Readme.php>

El proyecto de ejemplo “Dense Matrix Multiplication sample” se multiplican 2 matrices, dividiendo el proceso de multiplicación en tareas JPPF donde cada tarea multiplica una fila de la primer matriz por la de la segunda matriz. Este proyecto se construye principalmente de 4 clases: la clase MatrixRunner (que es la clase principal), la clase Matrix (la clase que representa a la matriz), la clase MatrixTask (que corresponde a la tarea JPPF a procesar) y la clase ExtMatrixTask (que acompaña a la clase MatrixTask).

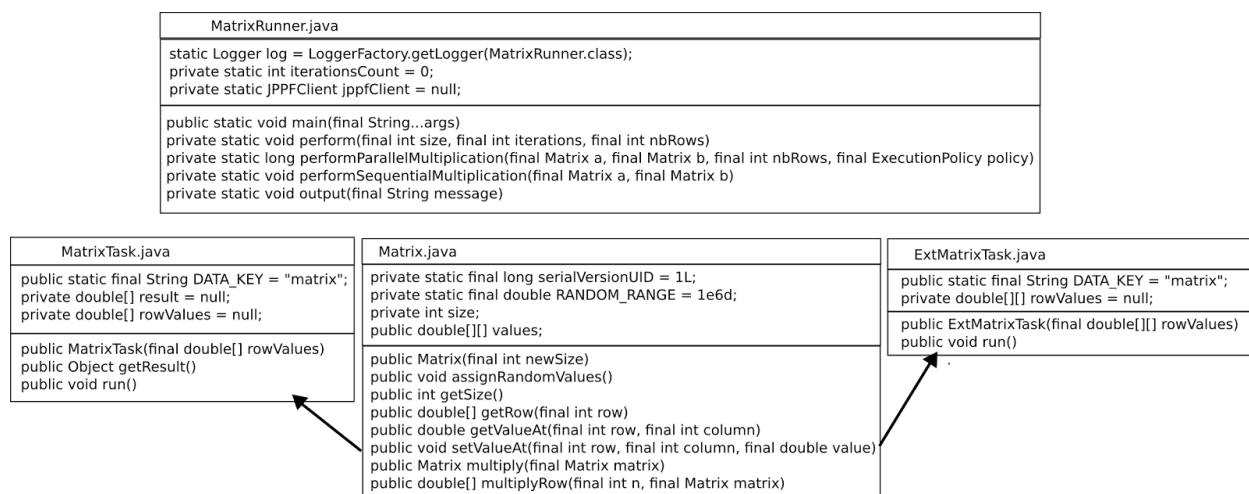


Figura 6.0.1.1 Diagrama de clase
Representa las clases correspondientes al ejemplo de multiplicación de Matrices

Desde la clase MatrixRunner se establece con la variable “size” el tamaño de las matrices, y con la variable “iterations” el número de iteraciones a realizar, posteriormente en el método “perform” se inicializan las matrices a las cuales se les asignan valores aleatorios mediante el método “Matrix.assignRandomValues()” que les da el valor aleatorio “RANDOM_RANGE * (2d * rand.nextDouble() - 1d)”. Después mediante el método performSequentialMultiplication “performSequentialMultiplication” se establece el tiempo aproximado que tardará la matriz en ser multiplicada y mediante el método “performSequentialMultiplication” se distribuirán las tareas JPPF en los diferentes nodos, midiendo el tiempo que consumió cada iteración de la multiplicación para devolverlo como resultado final. Para casos de ejemplo la figura 6.0.1.2 ilustra la forma de trabajar de “MatrixRunner.java”

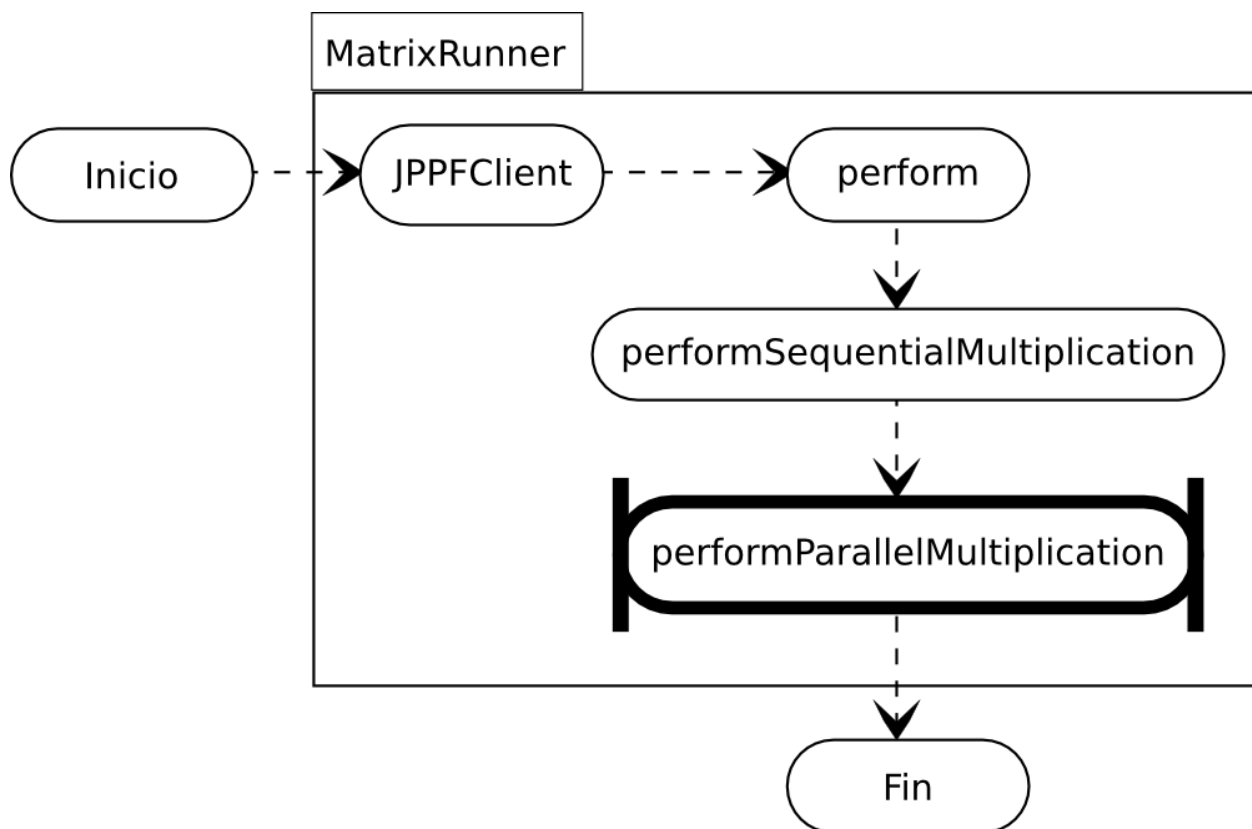


Figura 6.0.1.2 Diagrama de flujo

Ilustrar mejor la forma de trabajar del ejemplo de Matriz a continuación se muestra un diagrama de flujo correspondiente a la clase principal “MatrixRunner.java”, donde los ovalos resaltados corresponden a las tareas o inicio de las tareas a ser distribuidas.

Resultado de las pruebas de la aplicación Matrix

* Nota: El siguiente resultado expuesto es el mismo obtenido por la pantalla de Eclipse. Y cabe decir que el código original no implementa o manda un mensaje a ser expuesto en cada nodo donde se procesa una parte del trabajo.

Resultado:


```

log4j:WARN No appenders could be found for logger (org.jppf.utils.JPPFConfiguration).
log4j:WARN Please initialize the log4j system properly.
client process id: 5844, uuid: B438AE8D-FCED-96DE-2F94-84E0C63D2C22
_ * Corriendo demo de Matriz con tamaño = 300*300 para 10 iteraciones_
_ Cálculo secuencial realizado en 00:00:00.124_
[client: jppf_discovery-1-1 - ClassServer] Attempting connection to the class server at
192.168.0.111:11111
[client: jppf_discovery-1-1 - ClassServer] Reconnected to the class server
[client: jppf_discovery-1-1 - TaskServer] Attempting connection to the task server at
192.168.0.111:11111
[client: jppf_discovery-1-1 - TaskServer] Reconnected to the JPPF task server
_ Iteracion #1 hecho en 00:00:03.349_
_ Iteracion #2 hecho en 00:00:00.282_
_ Iteracion #3 hecho en 00:00:00.259_
_ Iteracion #4 hecho en 00:00:00.268_
_ Iteracion #5 hecho en 00:00:00.211_
_ Iteracion #6 hecho en 00:00:00.251_
_ Iteracion #7 hecho en 00:00:00.198_
_ Iteracion #8 hecho en 00:00:00.201_
_ Iteracion #9 hecho en 00:00:00.245_
_ Iteracion #10 hecho en 00:00:00.150_
_ Tiempo de iteración: 00:00:00.541_

```

Se hicieron pruebas con diferentes números de computadoras conectadas en una red y cada una con un nodo levantado.

Resultados de las pruebas:

* Para establecer el tiempo que tardó cada prueba se utilizó un cronómetro de reloj, por lo que pruebas futuras pueden variar en el tiempo.

* Se utilizó una red en constante uso para realizar estas pruebas.

Número de computadoras	Tiempo en segundos				
	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Prueba 5
7	13.44	10.10	9.88	10.35	10.07
6	9.00	9.10	8.97	9.32	8.44
5	7.64	7.59	8.13	8.21	8.47
3	6.16	6.84	6.31	5.74	5.95
2	7.32	7.13	6.72	6.78	6.85

1	6.72	5.87	5.75	7.77	5.52
---	------	------	------	------	------

A continuación las siguientes imágenes representan los cambios en el uso de la CPU y de la Red cuando era ejecutado el proyecto de prueba de multiplicación de matrices.

Sin ejecutar

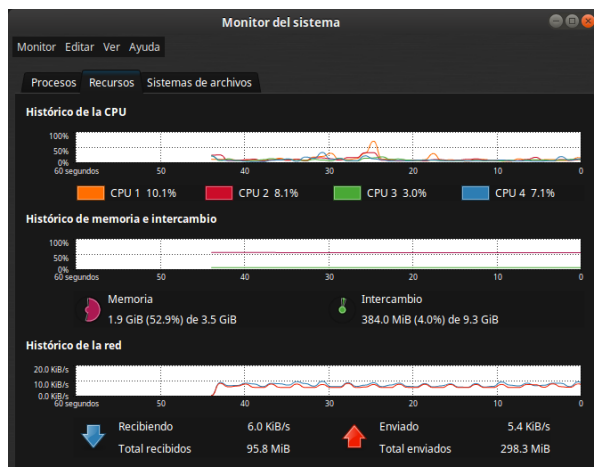


Figura 6.0.1.3

Corresponde al host del primer nodo y host servidor sin ejecutar el proyecto

En ejecución



Figura 6.0.1.4

Corresponde al host del primer nodo y host servidor cuando se ejecutaba el proyecto

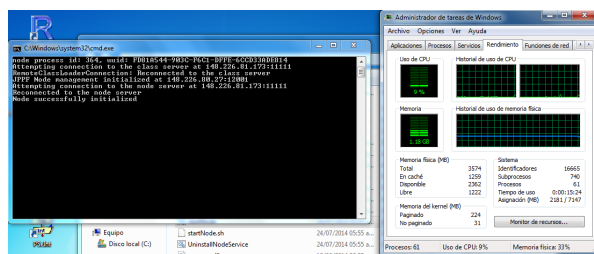


Figura 6.0.1.5

Corresponde al 2° host sin ejecutar el proyecto

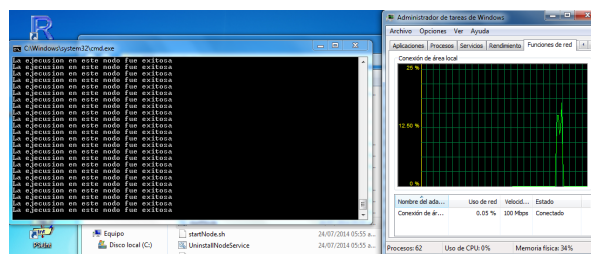


Figura 6.0.1.6

Corresponde al 2° host cuando se ejecutaba el proyecto

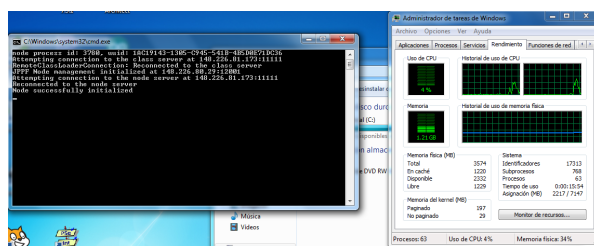


Figura 6.0.1.7

Corresponde al 3° host sin ejecutar el proyecto

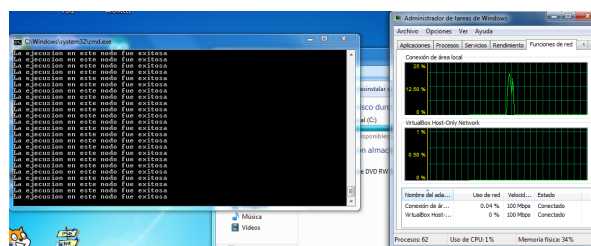


Figura 6.0.1.8

Corresponde al 3° host cuando se ejecutaba el proyecto

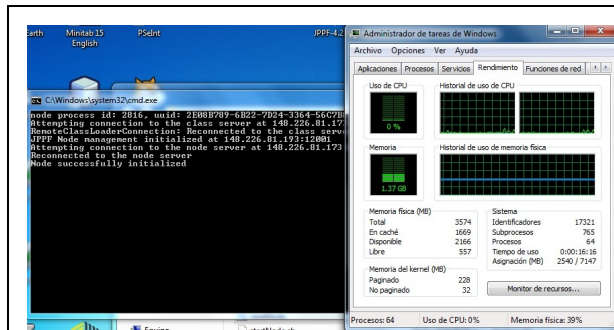


Figura 6.0.1.9
Corresponde al 4° host sin ejecutar el proyecto

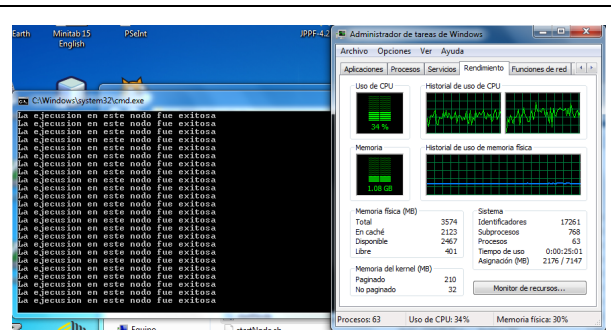


Figura 6.0.2.0
Corresponde al 4° host cuando se ejecutaba el proyecto

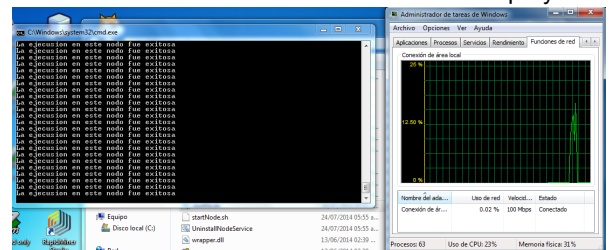


Figura 6.0.2.1
Corresponde al 4° host cuando se ejecutaba el proyecto

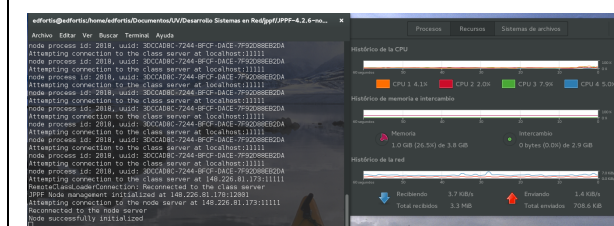


Figura 6.0.2.2
Corresponde al 5° host sin ejecutar el proyecto

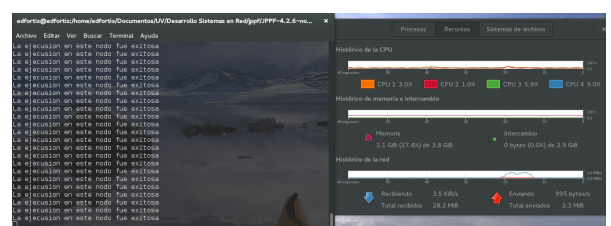


Figura 6.0.2.3
Corresponde al 5° host cuando se ejecutaba el proyecto

6.0.0. Ejemplo desde 0

Para crear un proyecto básico con JPPF es necesario importar las librerías de “/JPPF/JPPF-4.2.6-application-template/lib”

“/JPPF/JPPF-4.2.6-application-template/lib-src”

como lo muestra en la figura 7.0.1.1 del “application-template” el cual se se puede descargar desde la liga:

<http://sourceforge.net/projects/jppf-project/files/jppf-project/jppf%205.0/JPPF-5.0-application-template.zip/download>

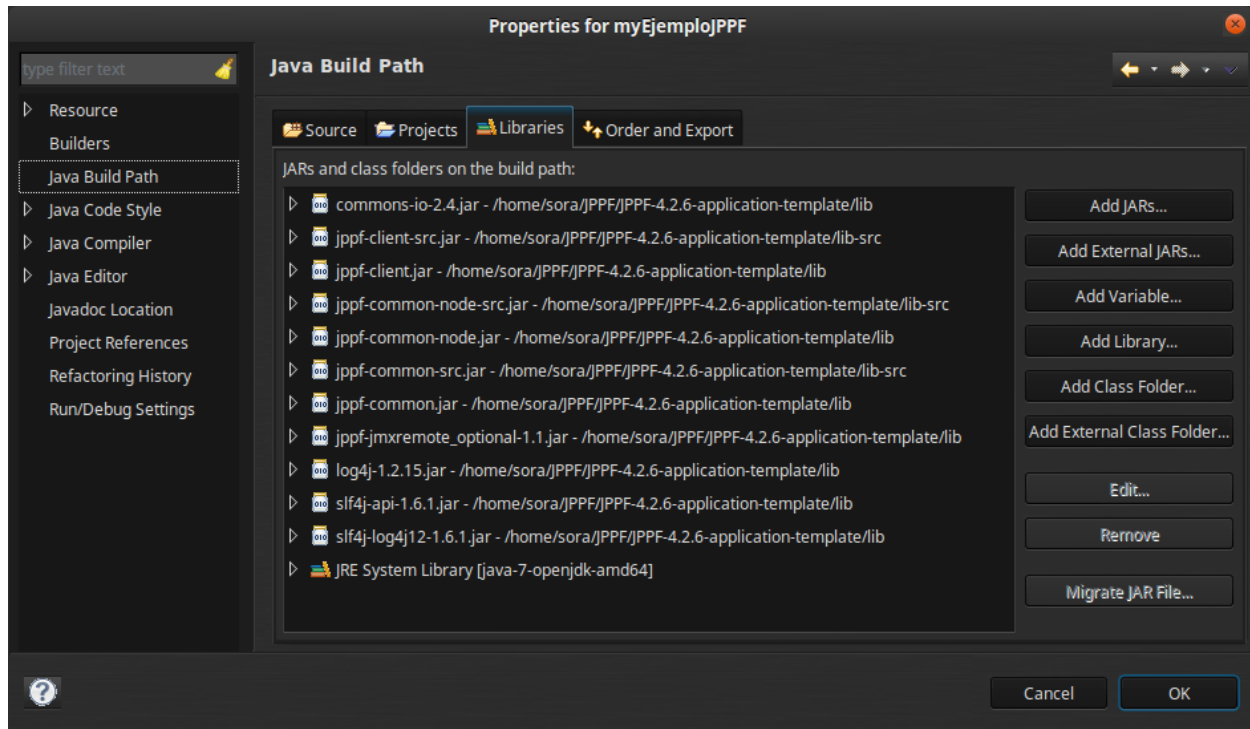


Figura 7.0.1.1.

Después es necesario crear una clase que importe “org.jppf.node.protocol.AbstractTask” y herede de “AbstractTask<String>”, implementando los métodos heredados.

```
public void run() {
    /*Aqui va el código de la tarea*/
    System.out.println("Este mensaje será visto en el nodo");
}
```

Teniendo la plantilla de la tarea ahora en una clase principal que importe “org.jppf.client.*” y cree un JPPFClient en un try{}catch(Exception e){}. Establece dentro del try{ el inicio de una clase trabajador que se encarga de crear y distribuir proceso o la carga de las tareas}

```
try (JPPFClient jppfClient = new JPPFClient()) {
    trabajadorJPPF trabajador = new trabajadorJPPF();
    trabajador.ejecutarTrabajos(jppfClient, numeroDeTrabajos);
}catch(Exception e){
}
```

Finalmente se crea una clase trabajador que se encargara de crear y distribuir las tareas en los nodos, la clase trabajador debe importar “org.jppf.client.*” y “org.jppf.node.protocol.Task”. La clase trabajador que a continuación se mostrará consta de 4 métodos, de las cuales 2 métodos construyen las tareas, 1 las distribuye en los diferentes nodos y un último método asegura su distribución.

```
public void ejecutarTrabajos(final JPPFClient jppfClient, final int numTrabajos) throws
Exception {
    Prepara, y empieza la ejecución de los trabajos(tareas)
}

public JPPFJob crearTrabajo(final String nombreTrabajo) throws Exception {
    Crea o inicializa un trabajo (tarea) a procesar en un nodo disponible
}

public void asegurarNumConexiones(final JPPFClient jppfClient, final int
numberOfConnections) throws Exception {
    Asegura que exista conexión con los nodos y prepara un previa distribución del
trabajo (tarea)
}

public synchronized void procesarResultadosDeEjecucion(final String
nombreTrabajo, final List<Task<?>> results) {
    Manda los trabajos (tareas) a cada nodo, los ejecuta y recibe los resultados los
cuales interpreta y muestra.
}
```

7.0.0. Conclusiones.

Nosotros consideramos que JPPF es una gran herramienta para la agilización de procesos, trabajos o tareas de una aplicación de java; es bastante útil y provechoso que al ser JPPF una herramienta de java, existe una gran documentación respecto a ella. De igual forma consideramos que los ejemplos presentados son útiles como provechosos para poder desarrollar un proyecto que utilice JPPF.

Creemos que el marco de trabajo que proporciona JPPF aporta un enfoque, y forma muy útil para desarrollar una aplicación, dado la posibilidad de distribuir procesos en paralelo, por lo cual agiliza a la propia aplicación. Sin embargo lamentamos que los

ejemplos más complejos de JPPF sea muy oscuros, de la misma forma lamentamos que no existan guía ejemplificadas para la instalación y configuración de JPPF como de todo el entorno o ambiente que este requiera. Aunque comprendemos que a pesar de ser una potente tecnología, no es una tecnología muy usada como conocida.

8.0.0. Referencias

<http://www.jppf.org>

<http://sourceforge.net/projects/jppf-project/files/jppf-project/jppf%205.0/JPPF-5.0-samples-pack.zip/download>

<http://www.jppf.org/samples-pack/MatrixMultiplication/Readme.php>

<http://sourceforge.net/projects/jppf-project/files/jppf-project/jppf%205.0/JPPF-5.0-application-template.zip/download>

<https://internetenunclic.wikispaces.com/grid>

Cambios

DAR

diapositiva

tabla aplicaciones ejemplos, agenda, imagenes arquitectura, justificar todo el texto,

desventajas, cambiar orden entre aplicaciones jppf y ventajas y desventajas (antes de aplicaciones)

antes de ejemplos jppf mas ejemplos de la API (con las clases importantes)

mas detalle en el ejemplo JPPF mostrar la matriz, como funciona la idea poner

administrador de sistema con las graficas y ver si cuando se corre sube el

procesamiento red y cpu

ejemplo con proyecto pequeño

introducción

proceso de instalación

Documento

justificar documento

Marcelo:

- *La introducción*
 - *Que se trabajara en el documento contextualizar*
 - *Que es JPPF*
 - *Marcar el problema y qué problema resuelven los grid*
 - *propuesta de resolución del problema*
 - *Describir el documento (de que trata)*
-

Julian:

- *La arquitectura*
 - *Agregar imagenes que describan la arquitectura*
 - *Figura con título y referencia en el texto.*
 - *Ventajas y Desventajas*
 - *Agregar desventajas*
-

Nota: No poner puntos a los títulos

Eduardo:

- *API*
 - *Poner y describir las clases más importantes*
 - *Ejemplos de aplicaciones (Casos de estudio)*
 - *Colocar como una tabla los ejemplos de aplicaciones*
-

Jorge:

- *Ejemplo de aplicación*
 - *Agregar explicacion de como está descrita la matriz, como se hacen las multiplicaciones.*
 - ** Nota: El código no es necesario*
 - *Solo mostrar los resultados de las pruebas, agregar mas pruebas y diferentes ejecuciones de las mismas.*
 - *Agregar referencia al sitio donde esta en codigo*
 - *Parte del grid*
 - *Mostrar los procesos y en red (Imagen del administrador de proceso)*
-

Alguien:

- *Ejemplo único desde 0*
 - *¿Como se construye el proyecto y que necesito?*
-

Jesus:

- *Conclusiones*
 - *Describir problemas encontrados al desarrollar el proyecto.*
- *Referencias*
 - *Agregar enlaces o fuentes de donde fue extraída la información*