Estrategias para la gestión del workflow administrativo en entornos GRID: Integración con JPPF

Trabajo Fin Grado Ingeniería Informática

Autor: José Herrera Hernández

Dirigido por: Francesc Guim Bernat

Valencia Enero - 2012

A mis padres que les hubiera gustado ver este documento
A mi mujer que le ha gustado mucho ver este documento
A mis hijos que les ha gustado volver a jugar con su padre

The benefits of writing software
in a well-designed language
far exceed the costs.
Dennis Ritchie

ÍNDICE

Índice de Capítulos

Índice de Figuras

Índice de Tablas

Parte I - Alcance del proyecto

1.- Descripción y objetivos

Este primer capítulo se dedica a exponer las ideas preliminares sobre el proyecto que se ha desarrollado así como sus objetivos iniciales. Se han esbozado los comienzos del proyecto que han permitido la creación de una estrategia adecuada para la presentación de este trabajo. Todo esto concluye con el desarrollo de la temporización y la planificación de las tareas a realizar.

1.1.- Descripción del proyecto

La idea central es estudiar los procesos productivos en los ambientes de la administración pública. Se desea aplicar las tecnologías GRID al tratamiento de los workflows de trabajo en los accesos intergubernamentales de información. Se evaluará una selección de los frameworks para entornos GRID existentes con el fin de seleccionar aquel que tenga las características más adecuadas para lograr los objetivos a realizar y que gradúe el esfuerzo del entorno a implementar.

Para el entorno seleccionado se realizará un estudio más profundo y se verificará su comportamiento dinámico en entornos de prueba. Una vez evaluado el correcto funcionamiento de las herramientas seleccionadas se codificará un prototipo para la ejecución de un gestor de procesos con el fin de simular los entornos productivos gubernamentales.

1.2.- Objetivos del proyecto

Se han definido los siguientes objetivos para la realización del proyecto:

- Realizar un estudio, en base al conocimiento empírico, de las características propias de los trabajos en los entornos gubernamentales.
- Completar un catálogo básico de servicios generales que sean susceptibles de ser trasladados a la tecnología GRID.
- Estudiar la metodología para la introducción de un sistema GRID en entornos gubernamentales con el propósito de optimizar los procesos de intercambio de información.
- Definir y diseñar estrategias para la introducción de las nuevas tendencias tecnológicas en los ambientes gubernamentales con el fin de mejorar los procesos de negocio y agilizar las transferencias de datos.

1.3.- Cómo surgió la idea

El tratamiento de los datos que se realiza en las administraciones públicas posee unas características que son comunes a muchos entornos. En un principio se investigó en este sentido en el documento [36], durante la realización del proyecto de final del Master en Software Libre (UOC-2010). En este artículo se definió la forma de intercambiar información entre administraciones dentro de un entorno de agentes móviles. Se llegó a codificar, en entornos multicapa J2EE, una aplicación de prueba que permitía el

tratamiento de los ficheros de intercambio de una forma eficiente. Aunque los resultados fueron limitados, se inició el camino para el estudio de la aplicación de nuevas tendencias en el tratamiento de expedientes, en base a la computación distribuida.

La pretensión de evolucionar los sistemas públicos a entornos más distribuidos me ha llevado a utilizar las tecnologías GRID en este proyecto.

No es extraño encontrar trabajos donde las administraciones públicas implementan o utilizan herramientas para permitir la gestión de los trabajos internos. También son comunes los trabajos que diseñan estrategias de trabajo para el e-government o Web de cara a los ciudadanos. Todos estos se basan, bien en entornos Cliente/Servidor o bien en tecnología multicapa J2EE. Por el momento no he encontrado documentación que se centre en la posibilidad de utilizar tecnología GRID en este tipo de entornos.

Después de conocer JPPF (*Java Parallel Procesing Framework*) como framework de desarrollo GRID y tras verificar los pocos trabajos que se encuentran disponibles, se propuso la utilización de un framework de similares características como plataforma para el desarrollo del este trabajo final de grado. La tecnología GRID es una herramienta que se adapta perfectamente a los entornos de trabajo heterogéneos.

Nos podemos preguntar ¿y porqué en los ambientes gubernamentales? En un principio es por el amplio conocimiento del que dispongo en este ámbito, debido a mis años de experiencia. Pero la principal razón es que no se han implementado soluciones en este campo puesto que la tendencia general es la realización de trabajos evolutivos en los sistemas existentes, manteniendo la tecnología ya en funcionamiento.

1.4.- Cómo se realizará el proyecto

En este punto se describirá cómo se va a realizar el proyecto y la planificación que se ha seguido durante el desarrollo. El trabajo lo podemos dividir en cinco fases y estas en tareas que quedan agrupadas de la siguiente forma:

1. Estudio de JPPF.
   1.1. Análisis de documentación que proporciona JPPF y otras fuentes de información.
   1.2. Estudio de alternativas y selección del Framework.
   1.3. Definición de una estructura básica del prototipo.
   1.4. Realización de un prototipo.
2. Estudio de trabajos gubernamentales.
   2.1. Realización de un catálogo de procesos.
   2.2. Estudio de los procesos.
   2.3. Definición de los objetivos de mejora.
3. Estudio de alternativas para la migración de los trabajos gubernamentales a sistemas GRID.
   3.1. Diseño de estrategias de ejecución alternativas.
   3.2. Implementación de la solución adquirida.
   3.3. Definir las pruebas de la solución de migración.
4. Realización de un prototipo de implementación.
   4.1. Definición de prototipo.
   4.2. Planificación de la realización de trabajos base.
   4.3. Implementación de los trabajos.
   4.4. Ejecución y pruebas de los trabajos tipo.
5. Realización de la documentación.
   5.1. Redacción de la memoria.
   5.2. Realización de la presentación.

1.4.1.- Descripción de las tareas a realizar

Las tareas a realizar serán las siguientes:

1. Estudio de JPPF (*Java Parallel Procesing Framework*). El objetivo principal de esta primera fase será analizar y evaluar el framework como herramienta para la realización del trabajo. Se comenzará por la búsqueda y estudio de la documentación existente sobre ella y se concluye con la construcción de un prototipo con el fin de conocer su funcionamiento y limitaciones.
   1.1. Estudio de la documentación. En este punto se realizará una búsqueda de toda la información disponible en la Web y artículos que proporcionen referencias al funcionamiento de la herramienta.
   1.2. Estudio de alternativas y selección del Framework. Se estudiaran alternativas posibles para la realización del proyecto y se seleccionará una para poder realizar los prototipos, evaluación e implementación.
   1.3. Definición estructura prototipo. La definición de la estructura básica del prototipo, resultado de conjugar la disponibilidad de hardware y de las diferentes partes que posee, para utilizar el framework.
   1.4. Realización del prototipo. Se construirá el prototipo y se pondrá en marcha con el objetivo de conocer su funcionamiento ante las diferentes incidencias de funcionamiento.

2. Estudio de trabajos gubernamentales. Se pretende diferenciar los tipos de trabajos que se realizan en un entorno administrativo inter-gubernamental con el fin de diseñar una gestión integral de estas tareas en un entorno GRID. Definir los criterios de mejora que deben lograrse.
   2.1. Catálogo de procesos. Se realizará una catalogación de los procesos mediante diferentes criterios para poder definir una taxonomía de los tipos de trabajos.
   2.2. Estudio de procesos. Se analizarán los procesos propios de las administraciones públicas y sus principales características.
   2.3. Definición de objetivos de mejora. Se definirán los objetivos de mejora que se deberán evaluar en cualquier solución que se proponga.

3. Estudio de estrategias GRID. En una primera parte, y con el conocimiento adquirido en las primeras fases del trabajo, se deberán evaluar las diferentes alternativas existentes para poder diseñar las soluciones propuestas en los puntos anteriores. Después se realizará una implementación y una prueba de los módulos.
   3.1. Se definirán las estrategias de ejecución para la realización de las tareas catalogadas en la fase anterior.
   3.2. Se realizará la implementación de un prototipo, con vistas a verificar la corrección de las hipótesis realizadas en el punto anterior.
   3.3. Se definirán las pruebas que se deben realizar en los modelos y se ejecutarán sobre los prototipos diseñados con el fin de obtener resultados medibles y evaluables.

4. Implementar el prototipo. Se realizará un prototipo para la ejecución de trabajos tipo y obtener medidas empíricas de las ejecuciones con diferentes estrategias. Esto obligará a efectuar una implementación utilizando el lenguaje de programación Java y las librerías propias de JPPF.
   4.1. Definición del prototipo. En esta tarea definiremos el prototipo de implementación de la estructura que queremos utilizar para la ejecución de los trabajos. Este deberá simular un entorno productivo de intercambio de información entre administraciones o entre administraciones y empresas privadas.
   4.2. Planificación de la realización de los trabajos base. Se debe diseñar lo que van a ser los procesos que se ejecutarán emulando el funcionamiento de los factores que intervienen en la ejecución en el tratamiento de datos.
   4.3. Diseño e implementación de los trabajos. Codificar las estructuras anteriormente diseñadas para que puedan cumplir los objetivos marcados dentro del framework seleccionado.
   4.4. Ejecución y pruebas de los trabajos tipo. Ejecución de los trabajos tipo seleccionados y obtención de datos de la ejecución.

5. Documentación. En esta última fase se realizarán los entregables que deben presentarse como solución al PFG.
   5.1. Redacción de la memoria. Depuración y mejora de la documentación final del proyecto.
   5.2. Redacción de la presentación. Realización de la presentación del proyecto.

1.4.2.- Planificación temporal

La temporización prevista ha sido la siguiente:

Tarea	Fecha Inicio	Fecha Fin
Estudio de JPPF	3/10/2011	18/10/2011
Estudio Documentación	3/10/2011	14/10/2011
Definición estructura prototipo	14/10/2011	18/10/2011
Estudio de alternativas y selección del Framework	3/10/2011	8/10/2011
Realización Prototipo	5/10/2011	18/10/2011
Estudio Trabajos Gubernamentales	13/10/2011	21/10/2011
Catálogo de Procesos	13/10/2011	20/10/2011
Estudio de Procesos	19/10/2011	20/10/2011
Definición objetivo mejora	20/10/2011	21/10/2011
Generar documentación	20/10/2011	21/10/2011
Alternativas GRID	21/10/2011	19/11/2011
Diseño de estrategias	21/10/2011	26/11/2011
Implementación de solución	26/10/2011	19/11/2011
Pruebas de la solución	14/11/2011	19/11/2011
Generar documentación	15/11/2011	19/11/2011
Prototipo de implementación	21/11/2011	23/12/2011
Definición de prototipo	21/11/2011	23/11/2011
Planificación de trabajos base	23/11/2011	25/11/2011
Diseño e implementación de trabajos base	25/11/2011	21/12/2011
Ejecución y pruebas	12/12/2011	23/12/2011
Conclusiones	21/12/2011	23/12/2011
Documentación	23/12/2011	13/01/2012
Realización de la memoria	23/12/2011	13/01/2012
Realización de la presentación	4/01/2012	13/01/2012

Tabla 1 - Temporización del proyecto

Los hitos previstos en la realización del proyecto son:

Hito	Fecha
PEC1	10/10/2011
PEC2	30/11/2011
Punto Control	03/01/2012
Presentación PFG	13/01/2012

Tabla 2 - Hitos del proyecto

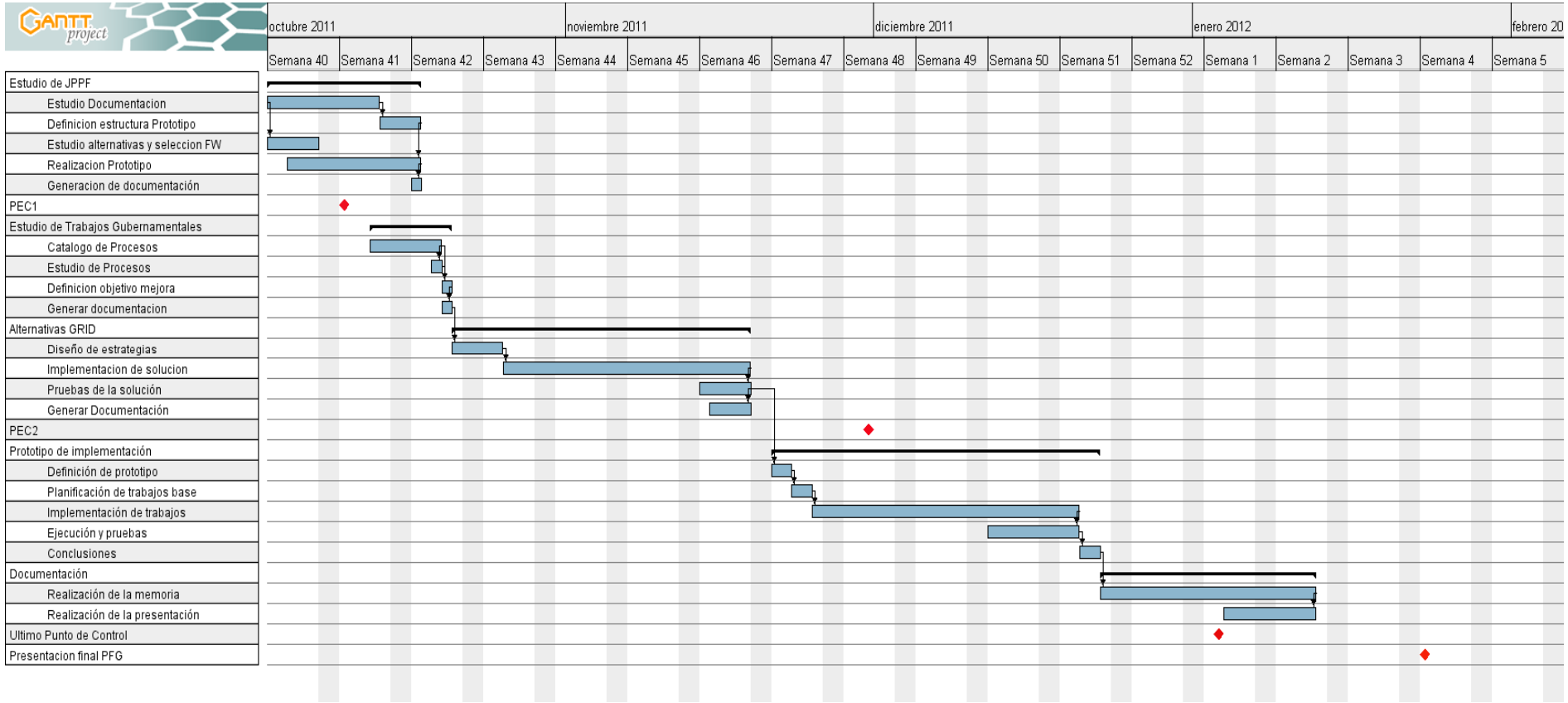


Figura 1 - Diagrama de Gantt planificado en el inicio del proyecto

Parte II - Tecnologías, herramientas y entorno de desarrollo

2 .- Computación distribuida: Alternativas

Durante este segundo capítulo se realizará un breve estudio de las tecnologías GRID con el objeto de poder enmarcar la herramienta de trabajo en el estado actual de la computación distribuida y de los entornos cluster. De esta forma se podrá afrontar, con unos conocimientos más amplios, los demás capítulos y las tareas asociadas a ellos.

Como resultado de este capítulo se seleccionará la herramienta que nos servirá para realizar las pruebas del prototipo que están previstas en el resto del trabajo.

2.1.- Tecnología GRID

Son numerosas las ventajas que presenta la tecnología GRID frente a tecnologías alternativas. La potencia de los computadores conectados en red es ilimitada, además de permitir una perfecta integración de dispositivos y sistemas heterogéneos. Las redes de computadores en GRID de gran tamaño son distribuidas y dinámicas. Proporcionan, como principal característica, un tiempo casi infinito de proceso y un almacenamiento sin limitación, pero también proporcionan acceso a gran variedad de dispositivos e instrumentos sin tener en cuenta su localización geográfica. Durante los últimos años esta tecnología se ha aplicado a diferentes áreas de las ciencias de la computación.

La documentación consultada está llena de referencias a dos ámbitos de la computación distribuida, la tecnología GRID y el *Cloud Computing*. En estos momentos no se plantea la alternativa de la utilización de otra tecnología de computación paralela que no sea GRID, lo que no quiere decir que una de las alternativas sea más valida que la otra.

Entre los beneficios tecnológicos que podemos encontrar tras implantar esta tecnología pueden encontrarse la integración de los entornos heterogéneos, facilidad de acceso a datos y trabajo colaborativo, creación de sistemas redundantes, mejora de la respuesta a variaciones de la demanda de servicios y nuevos requerimientos. Sobre el negocio se han identificado también beneficios como la mejora en la eficiencia operativa y la reducción de los gastos de capital acelerando los procesos de negocio.

Hay que tener un poco de precaución en la aplicación de estos sistemas con intención de mejorar los trabajos. La tecnología GRID no es una caja mágica donde poner cualquier aplicación y que se ejecute n veces más rápida sin tener que realizar ninguna inversión en hardware o software. No todas las aplicaciones pueden ejecutarse en un GRID con beneficios. Incluso existen aplicaciones que deben utilizar gran cantidad de recursos para su adaptación y lograr una mayor productividad. La configuración de los sistemas paralelos afecta enormemente a su productividad, disponibilidad y la seguridad de la infraestructura de la organización.

La tecnología GRID está en constante evolución. Por ello es necesario evaluar las nuevas herramientas que se presentan, así cómo estudiar la forma de adaptarlas a las necesidades específicas de los distintos entornos de operación.

Mediante un GRID podemos proporcionar diferentes tipos de servicios. Esta corresponde con la Taxonomía que presenta Buyya en [44]:

- **Servicios de Cómputo**. Esta utilidad está relacionada con la prestación de servicios seguros para la ejecución de trabajos en entornos distribuidos de forma individual o colectiva. La ejecución de aplicaciones en paralelo sería el objetivo final. Se puede poner como ejemplo la predicción atmosférica o los problemas de simulación nuclear.
- **Servicios de Datos**. Tienen que ver con proporcionar acceso seguro a bases de datos distribuidas y su gestión para proporcionar un almacenamiento escalable y acceso a grupos de datos. Estos pueden estar replicados, catalogados, e incluso con diferentes conjuntos de datos almacenados en diferentes ubicaciones para crear una ilusión de conjunto de almacenamiento. Un ejemplo de aplicación de esta categoría la tendríamos en el procesamiento de grandes conjuntos de datos en física de alta energía o bien en el acceso de bases de datos distribuidas para el análisis de las moléculas de diseño.
- **Servicios de aplicación**. Están relacionados con la gestión de aplicaciones y el acceso a distancia de bibliotecas de software de forma transparente. Las nuevas tecnologías, como pueden ser los Web Services, estarían enmarcados en este grupo.

Con la computación secuencial, basada en los sistemas centralizados, una organización debía hacerse cargo de todas las necesidades computacionales utilizando sus propios recursos, empleando grandes y costosos servidores con una potencia de cálculo muy grande. En estos sistemas podrían existir recursos sobrecargados junto a otros ociosos al mismo tiempo; además de tener una falta de escalabilidad y coste excesivo además de escasez de fiabilidad y robustez.

Aunque GRID y P2P parecen tener el mismo objetivo final (la organización coordinada de los recursos compartidos dentro de comunidades virtuales), se centran en distintas comunidades, por o cual tienen distintos requerimientos y siguen caminos evolutivos distintos. Mientras los sistemas GRID proporcionan servicios sofisticados a comunidades relativamente pequeñas y se centran en la integración de recursos muy potentes para proporcionar grandes calidades de servicio; los sistemas P2P tratan con muchos más participantes pero ofrecen servicios más limitados y especializados, y están menos preocupados por la calidad del servicio.

¿Qué conseguimos utilizando un GRID?

Cualquier empresa que tenga un pico de demanda o incluso una demanda cíclica estacional con épocas de mayor servicio y otras con valles en los que baja su utilización, tiene que disponer de una infraestructura informática que haga frente al umbral máximo de utilización. Si no, corre el riesgo de no poder atender las peticiones existentes, con la consiguiente pérdida de imagen y negocio.

Para cumplir una demanda desigual, podemos conseguir un grid en el que planifiquemos la utilización de recursos infrautilizados con el objetivo de dar cobertura a los procesos que en ese momento lo necesitan.

En el caso de que no contemos con una amplia base de ordenadores personales o de servicios que pueden desviarse para cubrir esa demanda, podemos recurrir a proveedores externos (*Grid Service Providers*) para que nos faciliten esa capacidad de forma puntual. Con ellos contaremos siempre con un pago por uso, es decir, tanto es lo utilizado tanto se paga. Así se transformará un coste fijo, que es la compra de nuestros equipos, en un coste variable, que es el alquiler de procesadores.

No quedaría completo este apartado si no se comentara algún caso de éxito de la propuesta GRID. Uno de los ejemplos donde se muestra la mejora que existe en la utilización de tecnologías GRID es el artículo [27]. En él se muestran los beneficios del uso de la tecnología GRID en el campo de la bioinformática. Este documento, que utiliza la infraestructura IRISGrid, explica cómo se ha conseguido una mejora del orden de 100 veces mayor que en centros de cómputo con capacidad limitada.

2.2.- Alternativas

En esta sección resaltaremos algunas de las herramientas GRID. Cualquiera de las descritas podría ser el objeto de este proyecto pero, por diferentes razones, no han sido las utilizadas. Existen numerosas alternativas en este campo pero no pueden ser analizadas todas e incluso su enumeración seria demasiado extensa. Al final, se han propuesto aquellas que tenían posibilidad de ser seleccionadas para formar parte del trabajo que se va ha realizar en los siguientes capítulos. También se ha incluido alguna propuesta, que aunque tenía escasas posibilidades, presentaba novedades o diferencias esenciales.

No se pretende realizar una profundización completa de las siguientes propuestas de framework, ya que a veces no es posible. Sin embargo, se proporcionará una visión global de estas herramientas, destacando sus bondades y enmarcándolas en el espacio de trabajo que cubren. Como modelos para el estudio se han seleccionado Globus Toolkit, Apache Hadoop, GigaSpace XAP, JPPF y gLite. Todos tienen alguna/s característica/s que los hacen especiales y que pasamos a describir.

2.2.1.- Globus Toolkit

Si tuviéramos que hablar de una única herramienta para el desarrollo GRID ésta sería Globus Toolkit (GT) [30]. Desarrollado por grupo de Carl Kesselman de la *University of Souther California* en Los Ángeles y el de Ian Foster en el *Argonne Nacional Laboratory*, es el ejemplo más popular de uso de un middleware de GRID llegando a considerarlo como el estándar "*de facto*" en *Grid Computing* por la comunidad internacional. En la actualidad se distribuye en su página web la versión 5.0.4. GT proporciona un grupo de componentes estándar que pueden dar soporte a gran variedad de sistemas personalizados sin tener que recurrir a complejos y costosos desarrollos a medida. No proporciona una solución "*ad hoc*" sino que presenta bloques constructivos y herramientas que pueden ser usados por integradores y desarrolladores para lograr sus fines.

Incluye herramientas como GRAM (*Globus Resource Allocation Manager*), GSI (*Grid Security Infrastructure*) que autentica a los usuarios y determina sus derechos, MDS (*Monitoring and Discovery Service*) que reúne información sobre los recursos que se están utilizando, GRIS (*Grid Resource Information Centre*) que dota de recursos de consulta sobre las configuraciones, capacidades y estado, GIIS (*Grid Index Information Service*) coordina los servicios GRID, GridFTP (*Grid File Transfer Protocol* [33]) provee un mecanismo de transferencia de datos de alto rendimiento, *Replica Catalog* proporciona información de las réplicas disponibles de unos datos determinados o bien *Replica Management System* que maneja el catálogo de réplicas.

Aunque son numerosas las razones por las que destacar esta herramienta - extensibilidad, potencia, capacidad y gran cantidad de proyectos aplicados - las dos razones por las que me gustaría destacar Globus Toolkit son: en primer lugar por la licencia libre con la que está publicado el código de esta herramienta y segundo lugar por que dispone de una bolsa de servicios que se aconseja utilizar, pero no es obligatorio. Como ejemplos de éxito de esta herramienta podemos destacar algunos que menciona en su propia página [22] cómo pueden ser la infraestructura utilizada para la simulación de terremotos del *Sourthern California Earthquake Center* o la simulación del fluido de la sangre en las arterias humanas en la *Brown University*.

Globus se distribuye con licencia *Globus Tookit Public License* (GTPL2) que podemos catalogarla como una licencia libre pero con algunas restricciones sobre condiciones de distribución, en el caso de desarrollos posteriores.

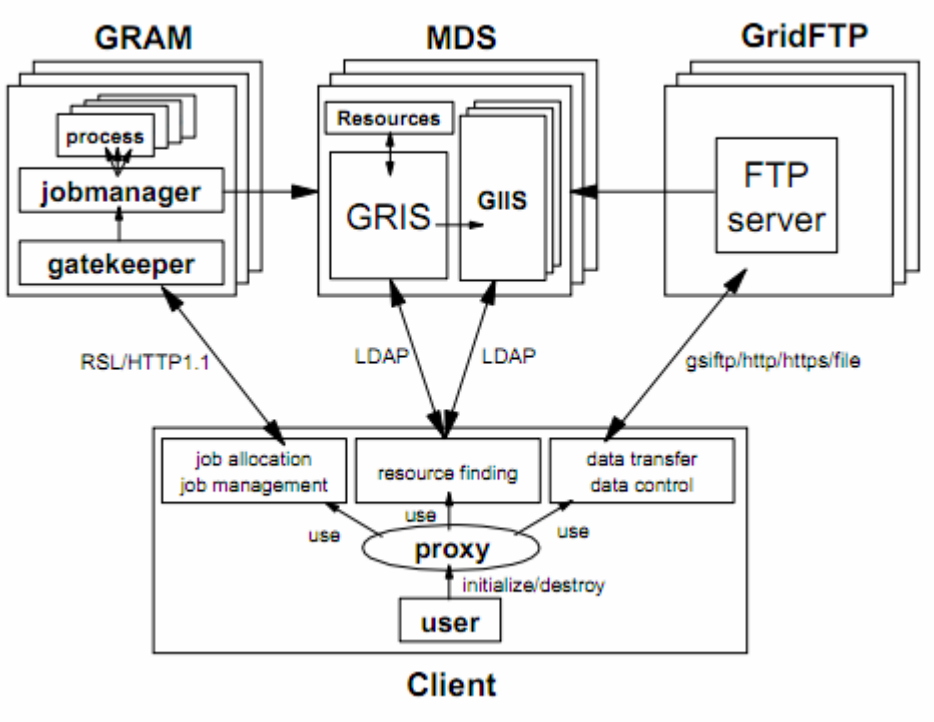


Figura 2 - Visión del Middleware Globus [23]

Existen otros dos componentes no mencionados hasta el momento que permiten el desarrollo de aplicaciones para este entorno:

- La API (*Application Programming Interface*) que está implementada principalmente en lenguaje C.
- *Developpers Toolkit.* La herramienta para el desarrollo rápido se conoce como CoG (*Commodity Grid*) que soporta entre otros los lenguajes y tecnologías siguientes: Java, Python, Perl, Web Services, CORBA, JSP o Mathlab.

Aunque en la selección del framework se analizan las razones por las que destacar uno sobre los demás con vistas a este trabajo, las principales razones técnicas que podemos remarcar en Globus para su posible eliminación de las propuestas son: complejidad, su difícil instalación y configuración, y que no existe ningún soporte para la migración de trabajos [18].

2.2.2.- Apache Hadoop

Hadoop [03] representa la propuesta de Apache para permitir el procesamiento distribuido de gran volumen de datos por medio de clusters de ordenadores, utilizando el paradigma de desarrollo Map/Reduce.

El proyecto incluye tres subproyectos:

- *Hadoop Distributed File System* (HDFS). Es la propuesta de sistema de ficheros distribuido.
- *Hadoop MapReduce*. Es el framework para el procesamiento distribuido en clusters.
- *Hadoop Common*: Las utilidades comunes a los dos proyectos anteriores.

Es utilizado por un gran número de empresas [03] en todo el mundo entre los que se pueden destacar: Facebook, Yahoo!, Twitter, Telefónica Research, Linkedin, eBay o Adobe. Este proyecto se considera como *Top Level Project* para Apache desde 2008. En la actualidad trabaja con volúmenes de datos del orden de Petabytes y clusters Linux de hasta 10.000 núcleos. Resulta una buena opción como sistema accesible, robusto, escalable y simple.

La idea principal de Hadoop es la de poder procesar grandes volúmenes de datos de forma compartida mediante su sistema de archivos.

El cluster común se compone de un nodo maestro y varios nodos esclavo. El maestro contiene el *jobtracker, tasktracker, namenode* y el *datanode*. El esclavo (*compute node*) contiene un *data node* y un *tascktracker*.

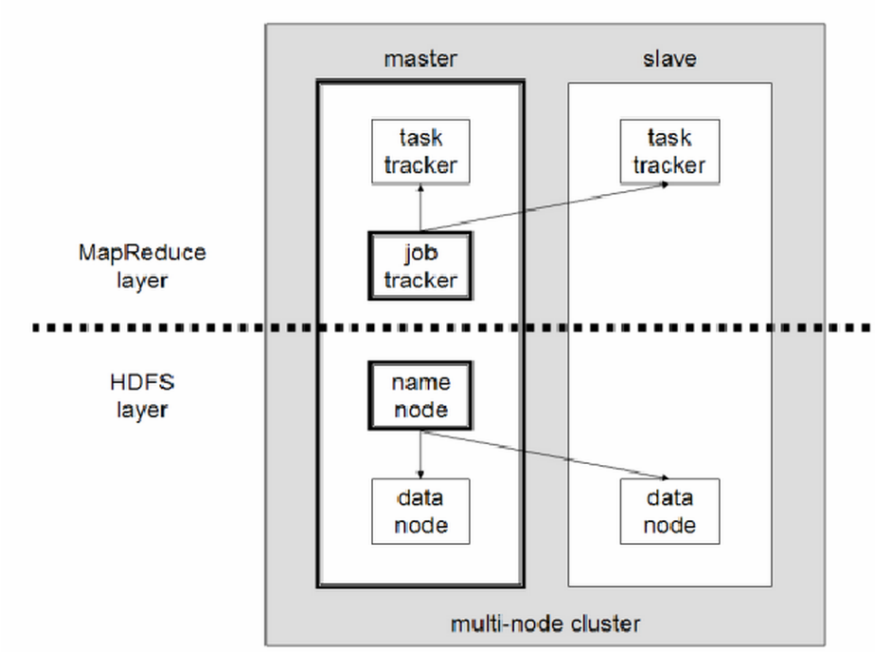


Figura 3 - Arquitectura Hadoop

Para el procesamiento de los trabajos se utiliza Map y Reduce [15]. Esta API es una de las contribuciones de Google en 2004 para la computación distribuida. Se realiza en dos fases:

- Fase Map. La función Map es la encargada de generar, desde un problema determinado, una lista de funciones, cada una en un dominio del problema diferente. Todas ellas juntas son equivalentes a la función original que utilizaba el dominio del problema en su totalidad.
- Fase Reduce. Una vez dividido el problema en problemas de inferior tamaño es necesario realizar la unión de los resultados de las funciones ejecutadas. En esta fase, el sistema será capaz de juntar los resultados producidos por las diferentes funciones Map y obtener un resultado final para el problema planteado.

El *Globus Resource Allocation Manager* (GRAM) [34] procesa las peticiones de recursos para la ejecución remota, localiza los recursos requeridos y monitoriza los trabajos activos. La monitorización y el descubrimiento de los servicios lo realiza MDS (*Monitoring and Discovery Service*) y la transferencia del fichero se realiza con SFTP (Secure FTP [60]). Para Hadoop la necesidad de compartir los ficheros se cubre con HDFS que está distribuido por las diferentes máquinas. El *Job Tracker* sería el GRAM con funcionalidad muy limitada. La parte de MDS está muy difuminada en la *MapReduce Layer* hasta tal punto que la lista de los nodos está situada en un fichero XML.

2.2.3.- GigaSpaces eXtreme Application Platform (XAP)

Aunque más desconocida en el entorno académico, no he querido descuidar las amplias y variadas propuestas de entornos GRID propietarias, aunque no resulta una propuesta que se tenga en cuenta para su evaluación final por su tipo de licencia. Existe una versión XAP de evaluación con limitación de memoria y servicios.

GigaSpaces [28] propone una gran variedad de servicios basados en la computación paralela y distribuida. Su producto estrella es XAP que está basada en Jini/JavaSpaces [48]. Se proponen diferentes herramientas, como un API para Map/Reduce, compatibilidad con las aplicaciones desarrolladas con arquitectura Java o con arquitectura de Aplicaciones Web. Soporta aplicaciones en Java, C++, .NET y un lenguaje de scripting propietario basado en Java.

Las diferentes capas que lo componen son las siguientes:

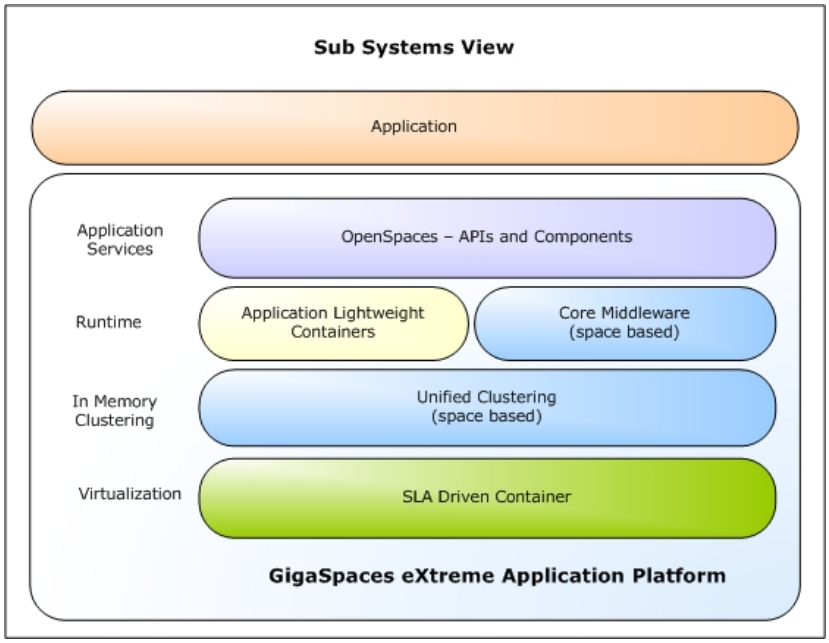


Figura 4 - Arquitectura Lógica XAP [28]

- *SLA-Driven Container* –compuesto por :
  - *Grid Service Manager* (GSM). Es el responsable del mantenimiento de los GSC.
  - *Grid Service Container* (GSC). Es el contenedor de las *Processing Units*. En general se puede ver como un nodo en el GRID. GSC comunica su estado a GSM.
- *Unified In-Memory Clustering*. Es la capa encargada de proporcionar los servicios para la escalabilidad, balanceo de carga y alta disponibilidad.
- *Core-Middleware*. Se compone de dos partes:
  - *In-Memory Data Grid*. Es la forma de almacenar la información. Totalmente transaccional, sigue las reglas ACID (*Atomicity, Concurrency, Isolation and Durability*).
  - *Message Grid*. Proporciona capacidades para el manejo de los mensajes, de uno a uno hasta varios a varios, permitiendo productores y consumidores asíncronos y la posibilidad de diseñar aplicaciones manejadas por eventos.
- *Lightweigh Applications Containers*. Proporcionan un entorno para la ejecución de aplicaciones a nivel de nodo. El GSC es el responsable de proporcionar capacidades GRID. Posee contenedores para la ejecución de Mule [65], C++, Microsoft .NET, Jetty [39] o Spring[59].
- *OpenSpaces-API and Components*. Está basada en Spring que es *open-source* y opera con un modelo POJO (*Plain Old Java Object*) [56] por lo que las aplicaciones pueden ser independientes del Framework.

Si lo comparamos con los dos firmwares anteriores debemos indicar su mayor similitud con Globus que con Hadoop por su estructura arquitectónica. GSM tendría su entidad equivalente en el GRAM de Globus, mientras que el *Unied I-Memory Clustering* sería solamente una parte de MDS mientras que los ficheros y los datos se pasan en vez de por SFTP o HDFS por medio de la *Message Grid*. Hay que destacar que también permite Map/Reduce como Hadoop.

2.2.4.- gLite

gLite [29] es un conjunto de componentes diseñado para facilitar la construcción de un GRID. Está producido por EGEE (*Enableing Grids for E-sciencE* [19]) que fue inaugurado por la división de tecnología del CERN en el 2004. En el 2011 se trasformó en el EGI (*European Grid Infraestructure*).

Este middleware es la reunión de varios trabajos realizados en otros proyectos, en particular Condor, Globus, LCG (*LHC Computing Grid* [46]) y VDT (*Virtual Data Toolkit* [63]). Proporciona al usuario servicios de alto nivel para el tratamiento paralelo de datos.

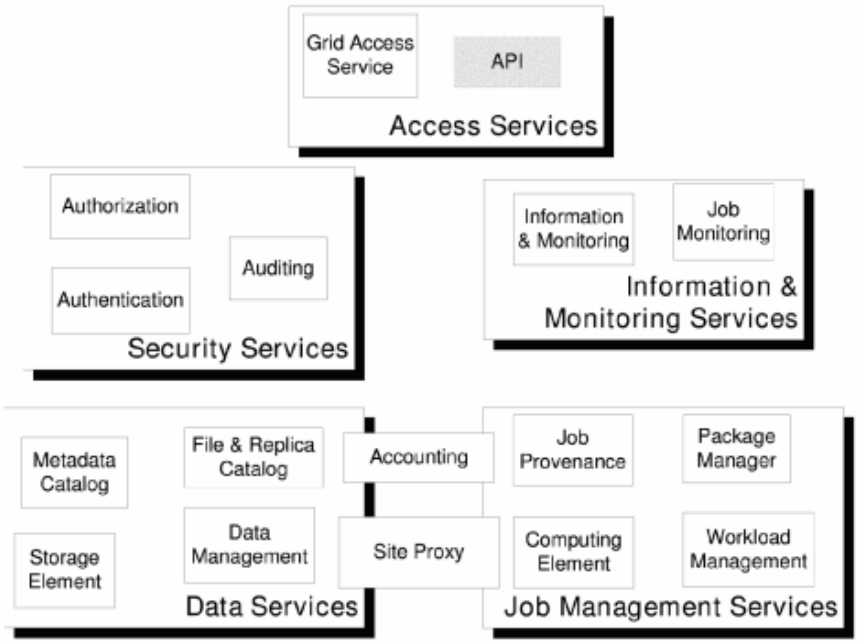


Figura 5 - Servicios gLite [45]

gLite es una implementación de la arquitectura SOA. Los servicios que presenta son los siguientes:

- Seguridad. Incorpora métodos para facilitar la seguridad en los desarrollos.
- Información y Monitorización. Proporciona mecanismos para la creación y utilización de información, incluyendo la monitorización del sistema.
- Gestión de trabajos. Este elemento se compone de
  - *Computing Element*. El principal elemento de este servicio que permite la virtualización de los recursos de ejecución.
  - WMS (*Workload Management System*) es un planificador para los trabajos.
  - El *Job Prevenance* (JP) permite almacenar información sobre los trabajos ejecutados con el fin de realizar inspecciones posteriores.
  - El *Package Manager* (PM) permite el despliegue dinámico de aplicaciones.
- Datos. Los principales servicios son:
  - *Storage Element* (SE) proporciona virtualización de los recursos almacenados.
  - *File & Replica Catalog Services*. Mantiene información sobre la localización de los datos y ciertos metadatos relacionados con estos (ckecksums y tamaño de los ficheros). Permite también las transferencias y movimientos, realizándolos de forma eficiente entre los SE.
  - *Data Management*. Toda la administración de datos se puede hacer sobre un único fichero o sobre un conjunto que se pueden ver como ficheros en un entorno Unix, permitiendo la navegación como en un sistema virtual, creando directorios y listando ficheros, etc.

Tal vez sea una de las propuestas más sólidas y cercanas ya que es la herramienta que se utiliza en el CERN (*Conseil Européen pour la Recherche Nucléaire*) para el tratamiento del gran volumen de información que manejan.

2.2.5.- Java Parallel Processing Framework

JPPF [42] representa una alternativa realizada en Java de un Framework completo para la implementación de tecnologías GRID. Creado por Laurent Cohen, se puede destacar por su simplicidad en la instalación y la facilidad en su configuración.

Es un Framework Java 100% que puede funcionar en cualquier sistema operativo que ejecuta una *Java Virtual Machine*. Se procuran herramientas para la monitorización y la gestión del entorno con estadísticas detalladas tanto por medio de GUI o por JPPF-API.

Se integra perfectamente con la arquitectura J2EE y con aquellos servidores que ofrecen estas funcionalidades: JBoss, Glassfish, IBM Websphere, Oracle Weblogic, Oracle OC4J, Apache Tomcat Server y con GigaSpaces eXtreme Application Platform.

Se componen de tres partes: el Nodo, el Driver (servidor) y la herramienta de monitorización. El intercambio de información entre los diferentes componentes se realiza mediante mensajes. No dispone de las mismas funcionalidades que otros frameworks ya que la seguridad, por ejemplo, solo la puede asegurar en las comunicaciones.

En el Anexo 4 se realiza una descripción más detallada de las funcionalidades y componentes de este framework que se han utilizado en el transcurso del proyecto.

2.3.- Selección del *FrameWork*

Ante todo remarcar que todas las alternativas son correctas para la realización de un proyecto de las características de este trabajo. De hecho, existen numerosos proyectos de trabajo en cada una de ellas que están en estos momentos en funcionamiento. Pero existen otros motivos para poder llegar a la selección que no solamente pasan por sus capacidades técnicas. Si sólo se evaluarán estas últimas, la selección hubiera sido otra.

Si deseamos evaluar las diferentes posibilidades debemos tener en cuenta criterios que pueden resultar externos a la implementación pero que resulten significativos. Por ello, uno de los primeros criterios ha sido la posibilidad de disponer de la herramienta con una licencia libre. Con esta premisa, XAP se descuelga de forma clara ya que, aunque no se duda de su calidad técnica, la imposibilidad de disponer de esta licencia imposibilita el trabajo en el proyecto.

Un segundo criterio es la posibilidad de realizar un trabajo sobre una plataforma poco estudiada. Si utilizamos el framework de trabajo Globus Toolkit tal vez entremos dentro de una de las herramientas más estudiadas sobre tecnologías GRID. Personalmente opino que, la utilización de una herramienta y su conocimiento, tal vez sea más beneficiosa, pero siempre se ha pensado que la posibilidad de utilizar otros frameworks menos estudiado enriquece el campo de la tecnología GRID. También tengamos en cuenta que posiblemente sea la herramienta más compleja de instalar y configurar.

La alternativa que planteo ha sido utilizar o bien Hadoop o bien JPPF. Llegado a este punto, se ha realizado un estudio más pormenorizado de las dos herramientas. He tenido que entrar un poco más a nivel técnico para poder solucionar la igualdad de las dos plataformas.

Desde un principio JPPF, ha parecido muy sencillo de instalar y poner un prototipo en funcionamiento no ha resultado muy costoso. Hadoop resulta ser una herramienta mucho más extendida que JPPF, y posee mucha más documentación y ejemplos, lo que puede resultar una gran ayuda en el caso de querer desarrollar código.

Finalmente Hadoop se basa más en la utilización del los sistemas de ficheros compartidos HDFS (*Hadoop Distributed File System*) [35] permitiendo la ejecución de tareas muy complejas mediante el modelo Map/Resume. Está diseñado especialmente para archivos muy grandes (Petabytes), con una

grabación y múltiples lecturas y hardware barato. No está diseñado para bajas latencias en acceso a datos (milisegundo), muchos archivos y muy pequeños o múltiples posibles escritores en posiciones arbitrarias.

Aunque los beneficios son muy grandes en grandes volúmenes de datos en caso de utilizar Hadoop, la propuesta de este trabajo, en el que se ha planteado el tratamiento de muchos ficheros pequeños, no es muy adecuado para esta plataforma.

El principal problema que se encuentra en gLite en su versión 3.1 es que solo es soportado por una arquitectura IA32 con una distribución de *Scientific Linux* [57]. Aunque es posible que en breve existan versiones para arquitecturas x86_64 y IA64 no está previsto que funcione en entornos Windows.

JPPF además de estar orientado a la realización de tareas complejas puede realizar tareas heterogéneas tratándolas de forma eficiente. Por ello he seleccionado este Framework, que aunque menos documentado y menos potente en sus capacidades, resulta más interesante en su estudio.

Esto no quiere decir que se considere JPPF como la mejor herramienta para el desarrollo en productivo, sino que se encuentra enmarcado dentro del grupo de frameworks poco documentados y que merece un estudio para poderlo comparar con las demás herramientas, más estudiadas por la comunidad científica.

Existen características como la extensibilidad o la escalabilidad que no se pueden comparar *"a priori"* ya que JPPF no posee ayuda en este sentido. Por lo tanto, como parte de este trabajo debería evaluarse la capacidad de ampliación de los frameworks de trabajo (ver sección 3.5).

Nos hemos dejado algunas de las plataformas GRID más destacadas como pueden ser UNICORE [62], Condor [13], ARC [07] o Boinc [08]. Creo que se ha dado un repaso entre algunas de las más destacadas para poder realizar una selección, desde las más estudiadas a las menos utilizadas y más modestas, desde las que tienen un cierto coste a las que son gratuitas.

A modo de recopilación de los apartados anteriores se muestra el cuadro resumen de las alternativas planteadas:

	Globus Toolkit	**Apache Hadoop**	**XAP**	**JPPF**	**gLite**
Categoría	Job/Data Schedule	Map/Reduce	Job/Data Schedule	Job/Data Schedule	Job/Data Schedule
Tipo	SOA Grid			Distributed master/worker and P2P	
HPC/HTC [1]	Supercomputación	HPC/HTC	Supercomputación	HPC/HTC	Supercomputación
Licencia	GTPL2	Apache 2.0	Con Copyright	Apache 2.0	Apache 2.0
Coste	Gratuito	Gratuito	Con Coste	Gratuito	Gratuito
SO	Linux	Linux	Windows, Linux y Solaris	Windows, Linux, Mac OS X, Solaris	Scientific Linux
Comunicación entre componentes	SOAP	Sistema de ficheros	Mensajes	RMI	SOAP
GUI	Web	Web	--	Java Swing, J2EE	Scientific Linux
API	C, Java, Python		Java, C++, .NET	Java	C
I/O	GridFTP[33]	HDFS [35]	Message Pasing	Herramienta Java	GridFTP[33]
Autenticación	X509	NO	AES, single sign-on	NO	X509
Otros	Standard "de facto"				
Entidad/Soporte	Globus Alliance	Yahoo!	GigaSpaces	Laurent Cohen	EGEE [19]

Tabla 3 - Resumen entornos GRID

.

[1] High performance/High-Throughput Computing

2.4.- Desarrollos similares

Después de la selección del entorno de desarrollo y todas las fases sucesivas del proyecto se ha encontrado y consultado numerosa documentación que se enumera con todo detalle en el apartado de bibliografía. En esta sección no se pretende enumerar herramientas que implementan tecnologías GRID sino proyectos que han supuesto una ayuda en la orientación de las diferentes fases del trabajo actual con la selección de la herramienta ya realizada. También se ha incluido en este apartado aquellos desarrollos en el framework de trabajo en el que el seleccionado destaca y que me han parecido interesantes.

Desearía destacar el trabajo [31] que ha permitido la orientación dentro de la plataforma JPPF para poder realizar los ejemplos, el documento [66] que utiliza el mismo Framework de desarrollo y el documento [26] que, aún teniendo otro enfoque diferente, pretende trabajar en ambientes GRID con problemas propios de las administraciones públicas.

Me parecen muy interesantes los desarrollos en el Laboratorio Nacional Sandia (Chicago). La infraestructura compuesta de 140 procesadores se utiliza, en la actualidad, para predicciones sísmicas en modelos terrestres 3D. También se utiliza en pequeños clusters de hasta 20 nodos para análisis estadístico, análisis lingüístico básico y algoritmos genéticos para redes neuronales. Lo consideran como un framework sencillo de utilizar, orientado especialmente a las tareas.

En cierta forma, la orientación que se desea para el proyecto es similar a los sistemas OMS (*Order Management System*). En nuestro caso la unidad principal de trabajo no es el pedido o encargo sino el expediente que se introduce en los sistemas por diferentes vías. Lo que resulta interesante es la evolución por diferentes estados de las órdenes, en similitud como lo hacen los sistemas informáticos de las administraciones públicas con los expedientes. De esta forma podemos encontrar numerosos procesos asociados como: los procesos financieros (tarjetas de crédito, pagos, etc.), procesamiento de pedidos (selección, verificación, impresión, etc.), análisis de los datos (verificación, incorporación, etc.) y otros.

3 .- JPPF (Java Parallel Processing Network)

En la realización de este capítulo se han estudiado las características propias del framework que se ha seleccionado, su clasificación y algunos de sus detalles técnicos. También se presentarán las pruebas y tests realizados en el GRID seleccionado como parte del estudio.

No pretende ser ni un manual exhaustivo ni una mera enumeración de las características técnicas del entorno seleccionado, que ya existe en su página Web. El principal motivo de la existencia de este bloque es destacar las peculiaridades propias de la herramienta que permitirán poner en marcha el entorno de estudio y pruebas o que puedan ser necesarias explicar para una mayor comprensión de posteriores capítulos.

Al mismo tiempo que se relacionan y comentan las características del entorno, se ha realizado una evaluación en condiciones controladas con el fin de obtener una idea más pormenorizada de las características y posibilidades que ofrece frente a las deseables en un sistema de este tipo.

3.1.- Introducción al Framework de desarrollo

El framework seleccionado para el desarrollo del proyecto ha sido JPPF desarrollado por Laurent Cohen [42] desde julio de 2007. La última versión publicada en la Web es la v2.5.3 [41].

Es un framework java que puede ayudar a los desarrolladores a codificar y ejecutar un programa java en paralelo. Permite trocear una aplicación java en partes independientes y ejecutarlas de forma paralela y autónoma en otras máquinas. Es muy útil en aplicaciones de cálculo intensivo que se pueden separar en diferentes secciones denominadas Tareas (*tasks*). Cada una de las tareas se compone de Trabajos (*jobs*) iguales o diferentes. Si queremos estudiar la posibilidad de introducir la gestión administrativa en un GRID será necesario en un principio no basarse en propuestas secuenciales ya operativas y diseñar el entorno libremente.

Se puede descargar bajo la licencia Apache v2.0. Como cualquier otra licencia de software libre, se permite al usuario del software la libertad de usarlo para cualquier propósito, modificarlo, y distribuir versiones modificadas.

3.2.- Características principales

JPPF funciona sobre cualquier sistema que soporte Java. No se requiere un sistema operativo en particular ya que puede ser instalado en todos los tipos de Unix, Linux, Windows, Mac OS y otros sistemas, como puede ser z/OS [68] y otros sistemas mainframe.

Los requisitos previos para su funcionamiento son:

- *Java Standard Edition* versión 1.5 o posterior, con la variable de entorno JAVA_HOME indicando la carpeta raíz de la instalación Java.
- Apache Ant versión 1.7.0 o posterior, con la variable de entorno ANT_HOME indicando la carpeta raíz de la instalación de Ant [06].
- Las inscripciones en la variable del sistemas PATH para JAVA_HOME/bin y ANT_HOME/bin

La totalidad del framework de desarrollo se puede descargar de la página Web de JPPF. El formato del fichero de descarga es *JPPF-xyz-<module-name>.zip* donde

X es el número de versión principal.

Y es el número menor de versión.

Z es el número de lanzamiento del parche. No aparece si no hay parche.

<*module-name*> es el nombre dado al módulo de instalación.

Para su instalación, después de las descargas, solo es necesario descomprimirlos y ejecutarlo.

De los frameworks de desarrollo sobre tecnologías GRID podemos destacar dos características que son obligatorias para poderlos considerar como tales:

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

En cualquiera de los dos casos JPPF está presente. En el primero se ofrece una abstracción de los problemas denominada Trabajos (Job) que está compuesta por numerosas Tareas (Task) que pueden ejecutarse de forma independiente. Para el segundo punto, JPPF proporciona un entorno de ejecución con servidores, nodos y herramientas de monitorización que permite soportar las abstracciones de Jobs y Tasks. Esta arquitectura permite la distribución de los diferentes trabajos que residen en los servidores a los distintos nodos que se ejecutan en máquinas diferentes e independientes.

El framework viene distribuido en ocho módulos diferentes. Cada uno de ellos tiene un objetivo diferenciado y pueden ser descargado y ejecutado de forma individual, dependiendo de las necesidades.

- *Driver.* Es el nombre que recibe el servidor de la infraestructura.
- *Node*. Es el paquete que contiene los ejecutables para la infraestructura de los nodos.
- *Console*. Es la GUI para la monitorización y administración del framework.
- *TCP Multiplexer.* Contiene los archivos necesarios para poner en funcionamiento nodos en localizaciones remotas.
- *J2EE Connector*. Contiene el paquete para el desarrollo de aplicaciones bajo la arquitectura J2EE.
- *GigaSpaces XAP Connector*. Conector para la aplicación GigaSpaces XAP.
- *Samples*. Ejemplo para la evaluación y pruebas.
- *Templates*. Ejemplo para el inicializarse en el desarrollo de aplicación en el GRID.

Para tener en funcionamiento una instalación mínima se debe ejecutar un *Driver* (servidor) y un *Node* (nodo) en una misma máquina o bien en dos máquinas conectadas en la misma subred TCP. El Server se convertirá en ese momento en el punto de acceso de todos los trabajos que se deseen ejecutar.

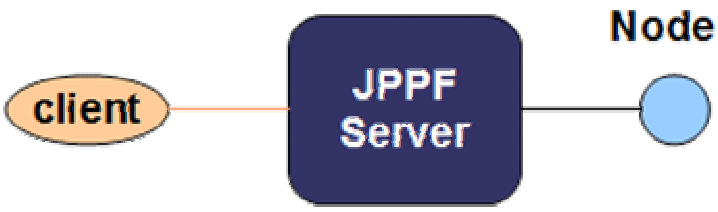


Figura 6 - Modelo básico de componentes de JPPF

En el Anexo 4 se han descrito con más detalle algunos elementos que serán utilizados durante las pruebas e implementación. Parece interesante comentar su funcionamiento y utilización dentro de la infraestructura como una parte más del estudio.

3.3.- Pruebas a realizar

Para la realización de tests del entorno debemos diseñar un grupo de pruebas lo suficientemente extensa que nos permita evaluar el correcto funcionamiento de las futuras aplicaciones que debemos poner en marcha. Al mismo tiempo adquiriremos la experiencia necesaria sobre el funcionamiento y manejo del framework.

Junto con los diferentes módulos descargables de los que se compone el framework, los desarrolladores han incluido un grupo de diecisiete ejemplos de funcionamiento y de herramientas de tests del entorno. Con la utilización de éstos, se ha logrado una rápida puesta a punto y unas pruebas realmente eficaces.

Los test se basarán, principalmente, en verificar y comprobar el correcto funcionamiento del framework seleccionado con respecto al balanceo de carga y la alta disponibilidad.

¿Cómo realizar una prueba de un sistema GRID?

Cuando se plantea la realización del proyecto, uno de los problemas que surgen es la evaluación de un entorno muy especializado como puede ser un entorno GRID. Algunos autores hablan de evaluar tres aspectos fundamentales: funcionalidad, fiabilidad y rendimiento [52]. En esta parte del estudio nos centraremos en la fiabilidad y el rendimiento que podemos obtener con la utilización del framework.

Los casos de evaluación del rendimiento se realizarán analizando el comportamiento de la plataforma de pruebas en saturación. Este estado se produce cuando el número de trabajos excede la capacidad de procesamiento del entorno.

Thomas Rings y otros [61] proponen un marco de pruebas especializado para los sistemas GRID del cual se extraen para nuestro caso los tipos siguientes:

- Test de arquitectura.
- Test de configuración.
- Test de aplicaciones.
- Test de selección y ejecución.

Las pruebas se realizarán con un entorno y con un framework muy concreto con el que se ha pretendido tener presente algunos de los escenarios más característicos de la operación del sistema seleccionado.

También extraídos del artículo de Thomas Rings y otros [61] se proponen, entre otras, las siguientes pruebas para verificar el correcto funcionamiento del entorno operativo:

- **Grupo A.- Pruebas de funcionamiento básico.**
  - A1.- Puesta en marcha de trabajos básicos con ejecución paralela en varios nodos.
  - A2.- Pruebas de puesta en funcionamiento de varios servidores (Drivers).
  - A3.- Mejora de la ejecución en paralelo.
- **Grupo B.- Evaluación de las características propias del GRID.**
  - B1.- En trabajos con gran utilización del sistema, verificar el balanceo de carga.
  - B2.- En trabajos extensos en tiempo, verificar el comportamiento del sistema ante caída de nodos.
  - B3.- En trabajos extensos, verificar el comportamiento del sistema ante la recuperación de nodos.
  - B4.- En trabajos extensos, verificar el comportamiento del sistema ante caídas de servidores (Drivers).
  - B5.- En trabajos extensos, verificar el comportamiento del sistema ante recuperación de servidores (Drivers).
- **Grupo C.- Pruebas de estrés del sistema (carga).**
  - C1.- Ejecución de trabajos muy intensivos en cómputo divididos en numerosos trabajos.
  - C2.- Ejecución de trabajos muy extensivos en tiempo divididos en numerosos trabajos.
  - C3.- Ejecución de trabajos mezclados, tanto en extensión de tiempo como en número de trabajos.
- **Grupo D.- Pruebas de escalabilidad del sistema.**
  - D1.- Verificar la mejora de tiempos con el funcionamiento de gran cantidad de nodos.
  - D2.- Verificar si existe mejora con la introducción de varios Drivers.
- **Grupo E.- Comportamiento del interface gráfico de usuario.**
  - E1.- Veracidad de los datos presentados en la pantalla.
  - E2.- Actualización de las pantallas.

3.4.- Estructura del prototipo

Para poder realizar las pruebas se ha diseñado una estructura que permita poner en marcha las diferentes funcionalidades que ofrece el framework evaluado. Las topologías Lógica y Física se presentan a continuación. Dependiendo de las pruebas a realizar se han utilizados unos nodos, otros o bien la infraestructura en su totalidad.

3.4.1.- Topología Lógica

La propuesta de topología lógica se compone de 7 máquinas con sistemas operativos Linux o Windows. Seis se encuentran en la misma LAN y una máquina fuera de la red de área local. La CPU externa se conecta a la red de área local mediante protocolo seguro ssh.

En cada uno de los equipos mostrados se instaló uno o varios nodos de procesamiento JPPF y en el equipo Lust se instaló además un Driver y la consola de monitorización. La aparición de más o menos drivers o bien la de threads/nodos en cada uno de los nodos dependerá de las pruebas a realizar más adelante.

Algunas de las máquinas que aparecen son máquinas virtuales que se han instalado sobre vmware Workstation 7.1.3 en un entorno Linux Debian. En la medida de lo posible no se utilizarán, pero en ocasiones se verificará el funcionamiento de los sistemas propuestos con nodos que se ejecutan sobre máquinas virtuales.

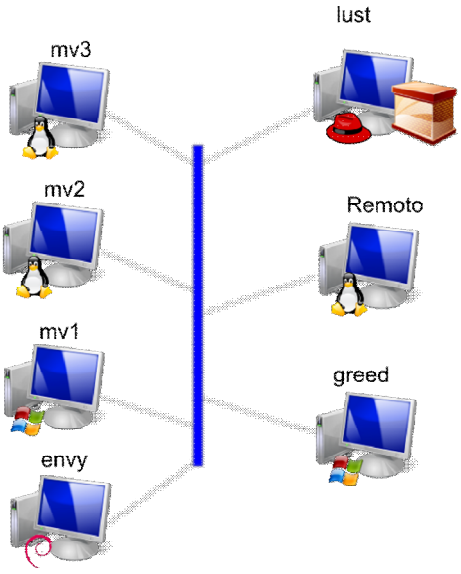


Figura 7 - Esquema lógico del prototipo

3.4.2.- Topología Física

En la topología física se puede observar la conexión de todos los computadores al mismo hub. La conexión de una de las máquinas, haciendo la función de gateway/firewall de cara al exterior, ha permitido añadir un nodo remoto mediante la conexión a Internet.

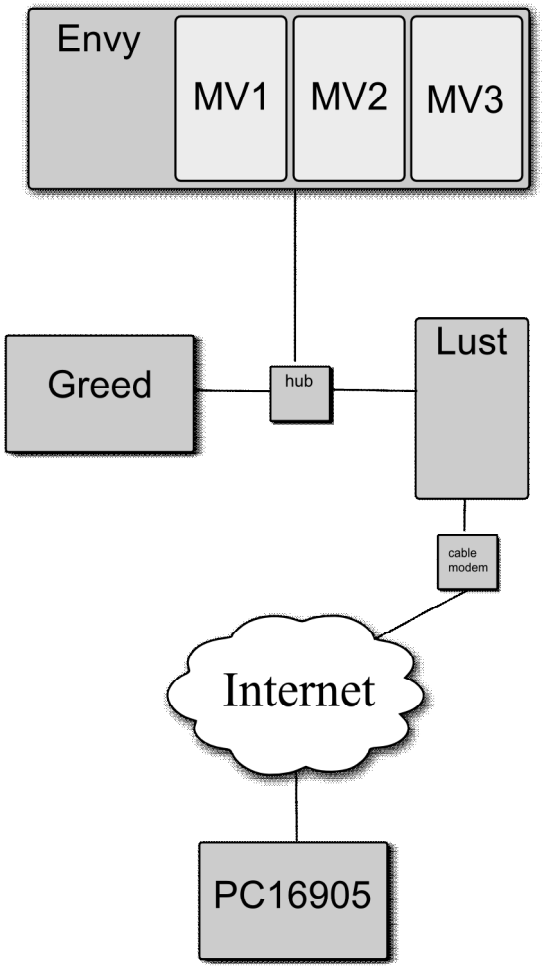


Figura 8 - Esquema físico del prototipo

Las características físicas de las máquinas utilizadas, no virtuales, son:

Nombre	CPU	Cores	Velocidad	Sistema Operativo	Memoria
Envy	Intel I5 760	4	2.8 Ghz	Debian 6.0.1 (Squeeze)	8 GB
Lust	Intel PIII	1	1 Ghz	Red Hat (2.4.20-openmosix2)	256 MB
Greed	Intel PIV	1	2 Ghz	Windows XP SP2	256 MB
Remoto (PC16905)	Intel Core 2 Duo E8400	2	3 Ghz	Windows XP SP3	2 GB

Tabla 4 - Características de las máquinas físicas del entorno de pruebas

3.5.- Ejecución y Resultados de las pruebas

En este apartado se comentará cómo se han puesto en marcha las pruebas que hemos diseñado en los apartados anteriores. Se describirá el método utilizado y aquellos detalles de configuración que aporten un conocimiento especial.

Aunque algunas de las pruebas que se han abordado no son especialmente significativas por el bajo número de nodos disponibles y la diferencia de potencia entre ellos, se han incluido los datos obtenidos de los nodos disponibles y se han comparado con algunos estudios sobre benchmarking existentes. Debemos tener en cuenta la diferencia potencia entre los diferentes nodos como causa de algunos valores extraños, pero la coexistencia entre diferentes máquinas forma parte de la naturaleza heterogénea de los Grid.

Con el fin de obtener resultados de los estados de ejecución, se ha puesto en marcha una consola de monitorización para observar el tratamiento que realiza el sistema de los jobs y tareas introducidos. También se han utilizado diferentes programas para verificar la utilización de la CPU en los nodos con el objetivo de comprobar la utilización tanto de procesador como de memoria.

Pruebas Grupo A.- Pruebas de funcionamiento básico.

En esta primera parte de las pruebas se pretende evaluar el funcionamiento básico de la infraestructura. Verificar que los nodos realizan las tareas, que es correcto el uso de varios drivers o que la ejecución en paralelo supone una mejora en el framework. Este test es fundamental para poder continuar considerando JPPF como una plataforma válida para el estudio.

(A1) Para este primer grupo de pruebas se han utilizado algunos de los ejemplos que proporciona la misma infraestructura. En concreto se han utilizado *MatrixMultiplication*, *Fractals* y *Nbody* para observar la distribución que se realiza de los trabajos en los diferentes nodos disponibles en el sistema (ver Anexo 4.9.1 para más detalles de las tareas).

Después de ejecutar numerosas veces los ejemplos, todas las pruebas de este punto han finalizado correctamente, pudiéndose observar una paralelización de los trabajos dentro de la consola del framework. El resultado ha sido satisfactorio.

(A2) Este segundo grupo de pruebas se ha efectuado con más de un Driver ejecutándose simultáneamente en diferentes equipos. Al lanzar los mismos programas que en el caso A1 no ha existido ninguna diferencia en el resultado. Por lo tanto, consideramos estas pruebas como satisfactorias.

(A3) Para realizar la prueba se puede observar el funcionamiento mejorado con el *MatrixMultiplication* con un tamaño de matriz de 500. En la siguiente captura de la ejecución se observa, en un primer paso, la multiplicación secuencial de las matrices realizada localmente y posteriormente se realizan varias ejecuciones en paralelo con el fin de evaluar el tiempo medio de mejora. En el caso capturado, la ejecución secuencial tiene una duración de 9.142 seg. mientras que la ejecución en el entorno GRID de pruebas tiene una duración de 2.459 seg.

```
[joseh@lust MatrixMultiplication]$ ./run.sh
Running Matrix demo with matrix size = 500*500 for 10 iterations
[client: driver-1] Attempting connection to the class server at lust:11111
```

```
[client: driver-1] Reconnected to the class server
[client: driver-1] Attempting connection to the JPPF task server at
lust:11112
[client: driver-1] Reconnected to the JPPF task server
Sequential computation performed in 00:00:09.142
Iteration #1 performed in 00:00:03.606
Iteration #2 performed in 00:00:03.180
Iteration #3 performed in 00:00:01.910
Iteration #4 performed in 00:00:02.509
Iteration #5 performed in 00:00:01.975
Iteration #6 performed in 00:00:02.354
Iteration #7 performed in 00:00:01.983
Iteration #8 performed in 00:00:02.524
Iteration #9 performed in 00:00:02.189
Iteration #10 performed in 00:00:02.365
Average iteration time: 00:00:02.459
```

Si lo observamos por los datos que proporcionan los gráficos configurables de la consola de JPPF se puede ver la ejecución de job (En el eje horizontal se muestra el tiempo y en el vertical el número de trabajos realizados).

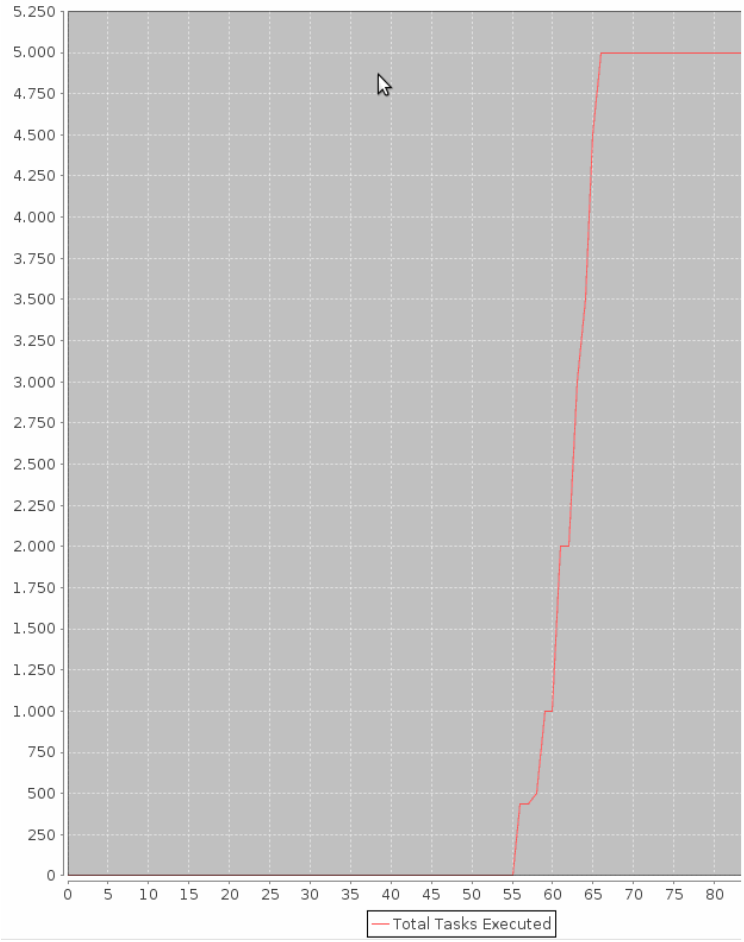


Figura 9 - Gráfico de ejecución de tareas

Ha existido una mejora de los trabajos, incluso en la pequeña infraestructura en la que se ha realizado y teniendo en cuenta la sobrecarga del framework. La mejora es de aproximadamente el **333%.**

Si realizamos una gráfica aumentando el número de threads hasta llegar al número total de cores disponibles obtendremos la siguiente distribución para el tiempo medio de ejecución:

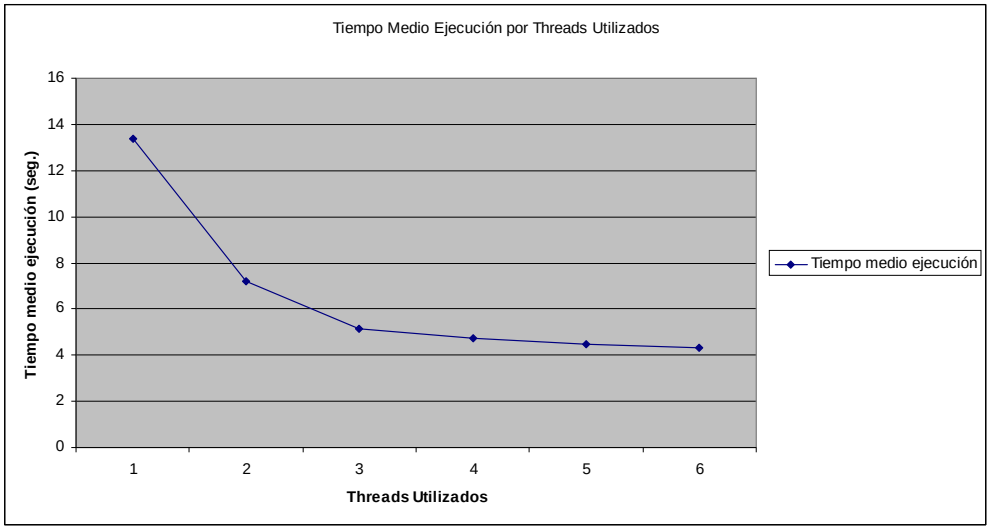


Figura 10 - Tiempo de ejecución medio en matriz de 1.000x1.000

Donde el eje X indica el número de threads que disponemos para la ejecución de la prueba y en el eje Y el total de segundos de la media de ejecución de una multiplicación de matrices de 1.000x1.000 un total de 10 veces. Con esta gráfica no se demuestra que con la utilización de más nodos exista una sobrecarga excesiva del framework sino que la utilización de máquinas con menos potencia no supone una mejora significativa para la ejecución de estas tareas intensivas en cálculo.

Si analizamos un poco más la secuencia de puesta en funcionamiento de los nodos, los cuatro primeros nodos que se han utilizado son los más potentes y se observa una tendencia asintótica con un tiempo mínimo de las mismas características que luego se acrecienta con la inclusión de los dos nodos menos potentes.

Pruebas Grupo B.- Evaluación de las características propias del GRID.

En este segundo grupo de pruebas se pretende evaluar algunas de las características más avanzadas, como el balanceo de carga o las capacidades de alta disponibilidad que se le suponen al framework por el hecho de ser un GRID. Para las pruebas de este grupo utilizaremos, de nuevo, la multiplicación de matrices usada en los apartados anteriores.

(B1) Para comprobar el balanceo de la carga entre los diferentes nodos de nuestra infraestructura se ha realizado la puesta en marcha de varios nodos y se ha lanzado varias veces la multiplicación de matrices. Se preveía un reparto de todas las cargas igualitario entre los diferentes nodos y los resultados obtenidos han sido diferentes.

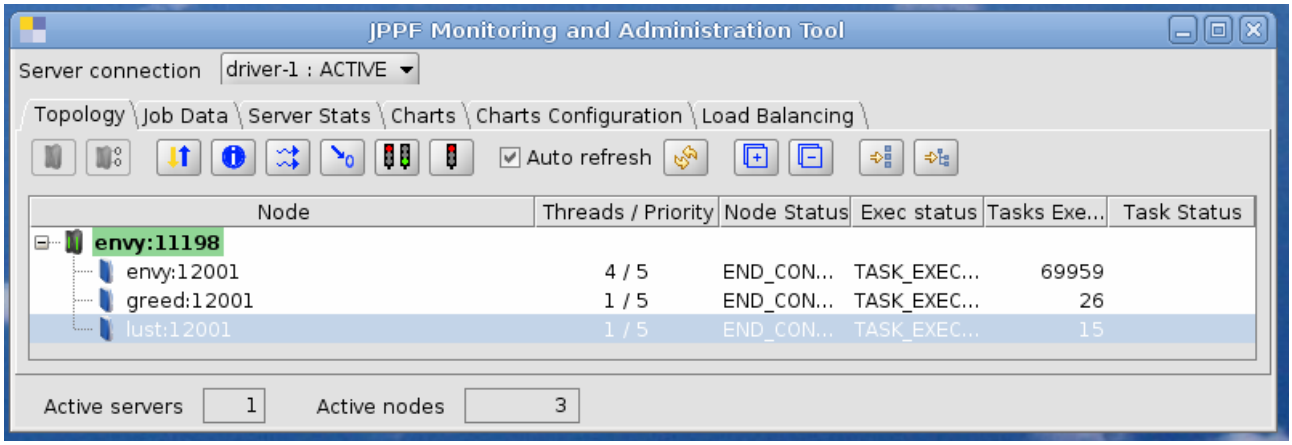


Figura 11 - Ejecuciones en los nodos

La captura demuestra que el balanceo de carga no es proporcional al número de nodos existentes sino que se intenta repartir la carga conforme se finalizan los trabajos realizados. De esta forma se observa claramente que el nodo envy es el más rápido (ver Tabla 4 - Características de las máquinas físicas del entorno de pruebas) y por lo tanto recibe la mayoría de los trabajos ya que los finaliza de forma más rápida. Lo que si se confirma es que el total de la suma de las tareas realizadas por el total de los nodos corresponde con el total de trabajos lanzados (69.959+26+15=70.000 correspondiente a 70 ejecuciones de una matriz de 1.000x1.000).

(B2 y B3) En ésta se lanzará un test que tenga una duración elevada en el tiempo. Se ha utilizado una multiplicación de matrices de 1.000x1.000 con una repetición de 100 veces. Debemos tener en cuenta que el objetivo de finalizar de forma correcta una determinada carga será prioritario con respecto al objetivo de obtener una elevada velocidad de ejecución [38]. En nuestro caso se puede simular realizando varias multiplicaciones de matrices de un tamaño elevado (1.000x1.000). En este escenario de ejecución se realizará la parada de los nodos de forma no programada y totalmente aleatoria.

Se ha verificado en 20 ocasiones la parada de los nodos y la recuperación de los mismos para seguir ejecutando los trabajos. En todos los casos la parada y recuperación ha sido correcta. El nodo puesto en marcha ha colaborado en la finalización de las tareas pendientes.

(B4) De la misma forma y con el mismo propósito de fiabilidad de nuestro entorno se han realizado pruebas de caídas de los drivers. Se ha dispuesto la infraestructura con tres servidores (Driver1, Driver2 y Driver3) que se han hecho caer de forma consecutiva en un periodo de tiempo.

El resultado ha sido correcto. Con la parada del primer Driver1 los nodos se han anunciado en el Driver2 y el proceso ha continuado correctamente. En el momento de la caída del Driver2 es cuando el Driver3 ha tomado la responsabilidad.

(B5) Para verificar el funcionamiento de la recuperación de los Drivers ante las incidencias de recuperación de operación se ha utilizado la situación final del caso B4.

Primero se ha intentado recuperar el Driver2 pero el resultado ha sido que los nodos no utilizan el nuevo Driver2, cuestión que parece normal. El problema que ha surgido es que al desaparecer el Driver 3 los nodos tampoco se conectan al Driver2 recuperado y la infraestructura deja de funcionar.

Este comportamiento erróneo no se ha alterado con la realización de modificaciones en los ficheros de configuración para indicar los Drivers que son utilizables.

Pruebas Grupo C.- **Pruebas de estrés del sistema (carga)**.

En esta fase de las pruebas se pretende evaluar la capacidad del sistema para realizar tareas de forma continuada y en situaciones de carga continua.

Como se ha comentado anteriormente, las pruebas se han realizado en una situación de saturación (ver como ejemplo Figura 19 - Procesos con conexión a la infraestructura). Para llegar a esta situación, todos los trabajos se han lanzado de forma simultánea con la única diferencia del tiempo que ha tardado en lanzar el proceso secuencial.

(C1) En este grupo de pruebas se utilizarán 4 procesos de multiplicación de matrices de 1.000x1.000 repetidas 5 veces. De esta forma se aumentará la complejidad de la ejecución de los trabajos y siendo un número reducido de tareas.

Ya que se produce la finalización de los trabajos no vamos a centrarnos en el tiempo de ejecución que parece que es correcto, según las pruebas básicas, sino que nos centraremos en cómo los trabajos están presentes dentro del sistema.

En este primer caso la cola de los trabajos aumenta al mismo tiempo que aumenta el tiempo que los trabajos están en el sistema. Por ello, se pueden ver los dos gráficos siguientes:

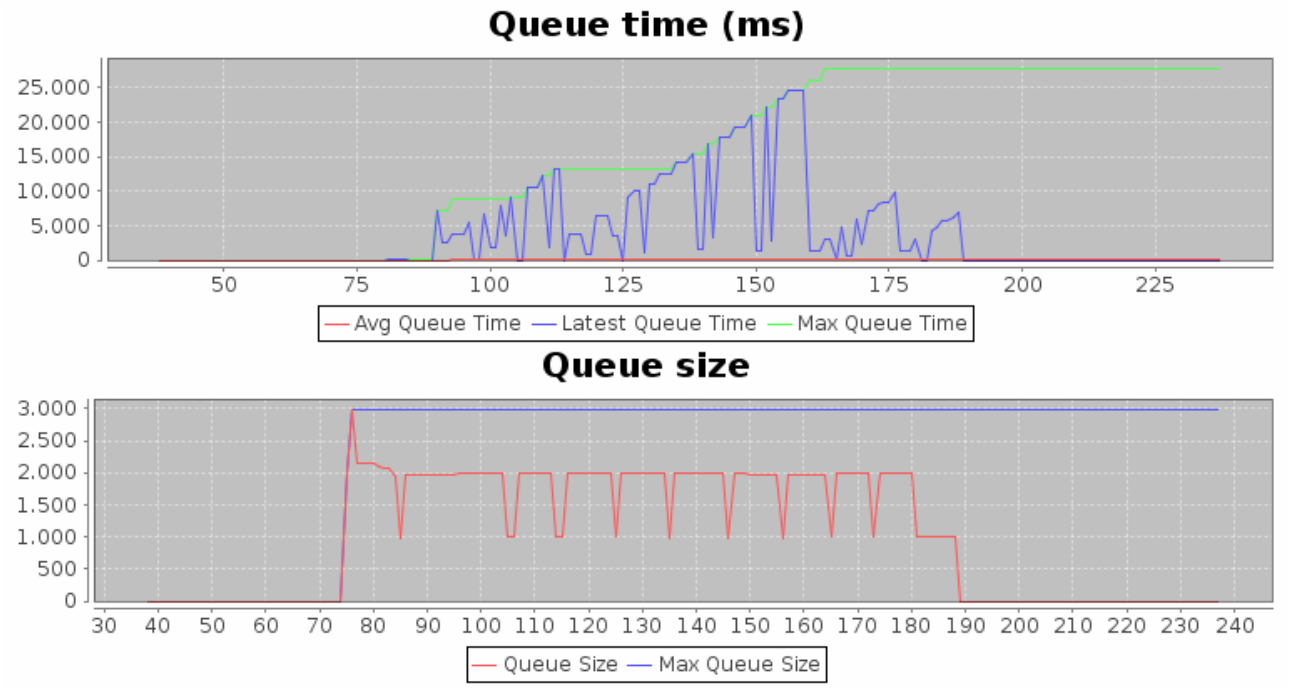


Figura 12 - Cola de ejecución trabajos de cálculo elevado

(C2) También verificaremos el comportamiento con numerosos trabajos mediante la utilización de otra prueba con la misma multiplicación de matrices. En este caso el número de tareas ha aumentado considerablemente pero son más simples (matriz de 100x100 y 50 interacciones). Se han lanzado 20 procesos en paralelo para llegar a la situación de saturación. Se puede ver lo mismo que en el caso C1, que existen algunos trabajos que esperan en la cola y que el tamaño de la cola aumenta.

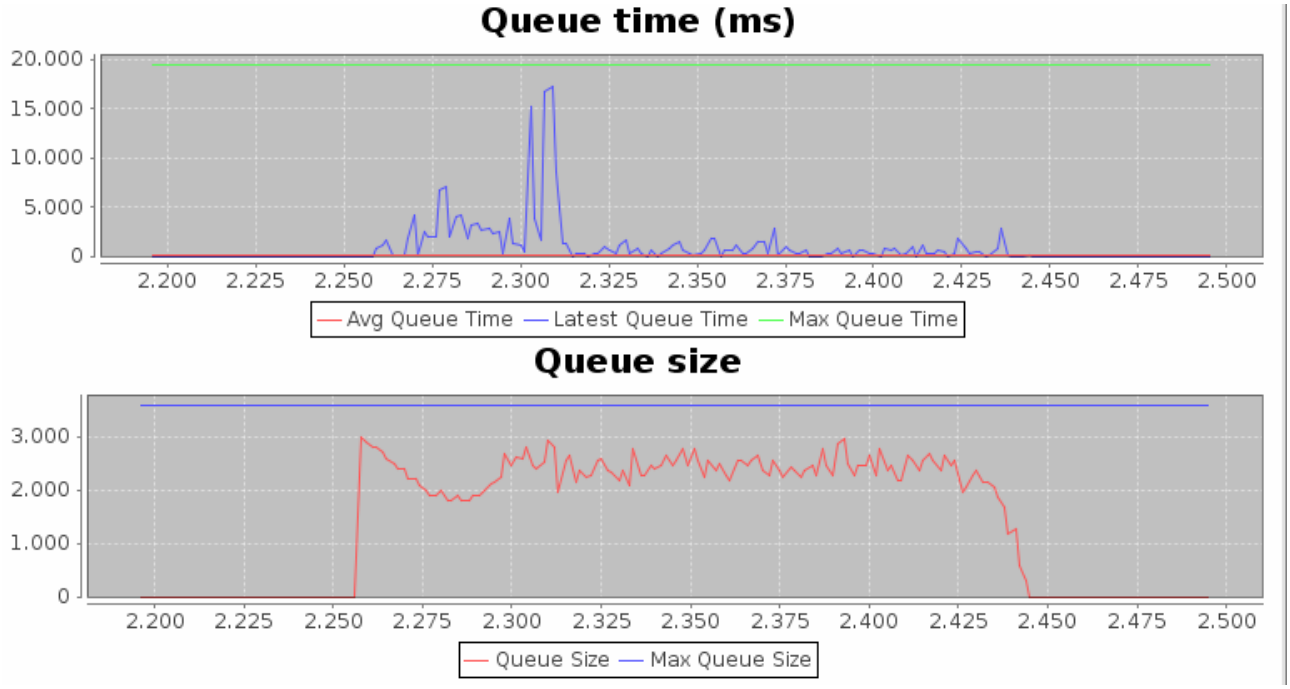


Figura 13 - Cola para la ejecución de gran número de trabajos

La introducción de los trabajos dentro del sistema es más lenta que la finalización de los mismos. Por ello, el tamaño de la cola se mantiene constante y sin embargo el tiempo de permanencia de los trabajos en esta cola es muy bajo.

(C3) Por último intentaremos realizar una prueba mezclando varios tipos de trabajos para que compitan en el prototipo. En este caso se utilizará la multiplicación de matrices, la ejecución de *N-body* y la generación de fractales en varias instancias para conseguir una utilización máxima de las CPU's de los sistemas utilizados. El resultado es que los procesos se reparten dentro de la infraestructura como se esperaba sin que ninguno quede sin servicio.

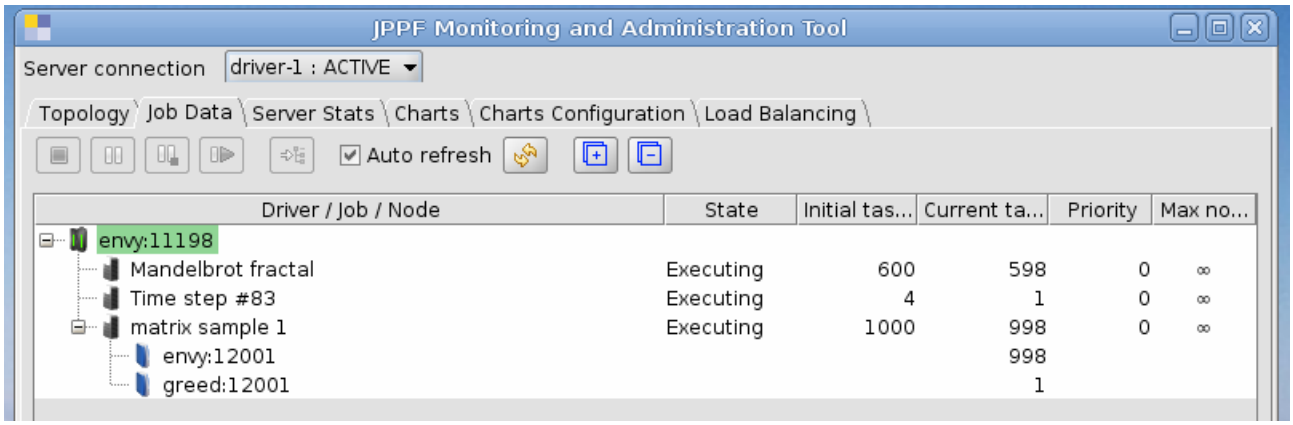


Figura 14 - Ejecución de trabajos variados

Pruebas Grupo D.- Pruebas de escalabilidad del sistema.

En este grupo de pruebas se intentará verificar la escalabilidad del entorno verificando los tiempos de ejecución de las aplicaciones con la aparición de varios nodos. Debemos tener en cuenta las limitaciones de hardware pero también que la introducción de un único nodo no hace que el procesador sea utilizado al 100%. Esto da cierto margen para poder aumentar el número de nodos, aunque sea en el mismo hardware utilizado hasta ahora.

(D1) En esta parte verificaremos que la existencia de varios nodos permite la ejecución más rápida de las aplicaciones, siempre que no se llegue a la utilización máxima de los procesadores del hardware utilizado.

Para verificar esta situación introduciremos un nodo en cada una de las máquinas y calcularemos el tiempo medio de ejecución del programa de multiplicación de matrices con un tamaño de 1.000.

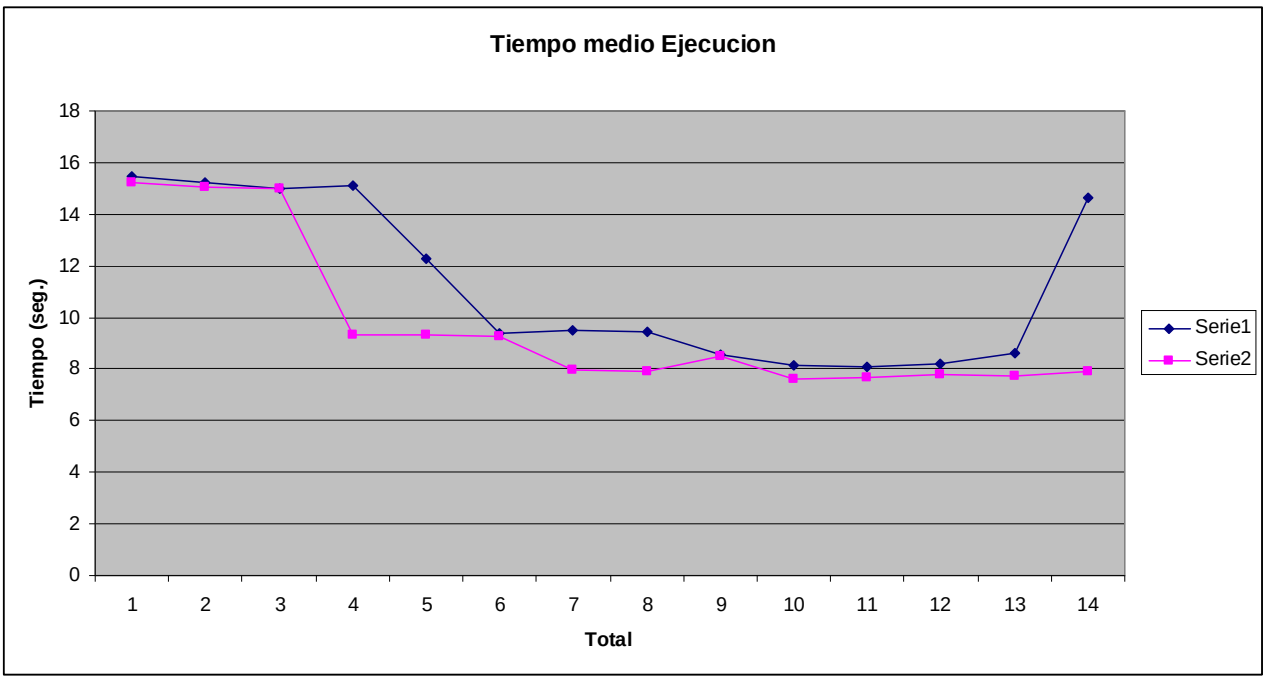


Figura 15 - Tiempo medio ejecución por nodo y Thread

La Serie1 corresponde la introducción de nodos en la infraestructura repartidos proporcionalmente por cada uno de los ordenadores que la componen. La Serie 2 corresponde al aumento, también proporcional, en cada uno de los nodos de los ordenadores, de la infraestructura de los threads que se ejecutan dentro de un único nodo por máquina.

Analizando la Serie1 podemos observar que el aumento de los nodos ejecutándose en la infraestructura no tiene ningún efecto sobre la ejecución a partir del 6º o 7º. Es destacable que si aumentamos el número de nodos más allá de los aconsejables el resultado es que la infraestructura se ralentiza.

Si analizamos la Serie2 vemos un resultado similar a la Serie1 pero la ralentización de la infraestructura no se ha observado con el mismo número de threads.

(D2). Para intentar verificar si existe alguna mejora en la existencia de varios Drivers se pondrán en marcha los tres Drivers del caso B4 procurando que los nodos se repartan proporcionalmente en cada uno de ellos. Con la puesta en marcha del primer driver se ejecutará la multiplicación de matrices de 1.000x1.000 con 10 repeticiones y se comprobarán los tiempos con cada uno de los drivers.

A diferencia de la prueba que se realiza en B1, se ha forzado que los nodos se conecten obligatoriamente a alguno de los Drivers mediante la modificación del fichero de configuración. Una captura de la distribución realizada de los nodos ha sido la siguiente:

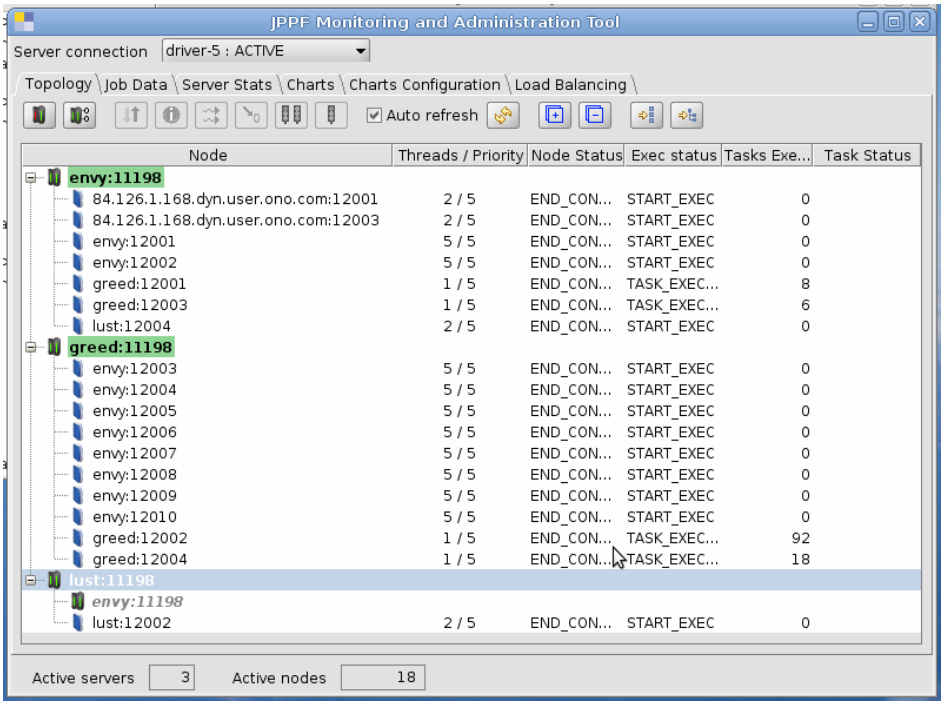


Figura 16 - Disposición de los nodos durante las pruebas

Los tiempos obtenidos de media con los diferentes nodos para la ejecución de una matriz de 1.000x1.000 son:

Total Drivers en ejecución	Tiempo de ejecución obtenido
1	10.232 seg.
2	47.212 seg.
3	49.212 seg.

Tabla 5 - Tiempo de ejecución con Drivers.

Todo indica que la aparición de los Driver añade una sobrecarga a la ejecución de los nodos elevada y que la conexión de los nodos a un único Driver es preferible a una situación de reparto de los nodos por los drivers. Esta sobrecarga de la infraestructura no es deseable.

Pruebas Grupo E.- Comportamiento del interface gráfico de usuario.

Este último grupo de pruebas se centra en asegurar el correcto funcionamiento de la consola de monitorización de JPPF. Algunas de las cuestiones que verificaremos ya se han utilizado en los tests realizados en las secciones anteriores.

(E1) En este punto se verificarán los datos que aparecen en la pantalla sobre la ejecución de jobs y tareas de las pestañas *Topology* y *JobData* y que tienen su correspondencia con la topología de la infraestructura y las cargas de trabajos que se realizan, respectivamente.

Para contrastar el correcto funcionamiento de la pestaña *Topology* nos centraremos en el número de tareas realizadas. Ejecutaremos el proceso de multiplicación de matrices con los datos de los nodos y Drivers puestos a cero, después de la ejecución deberá ser correcta la columna *Task Executed*. Este análisis ya se ha realizado en el punto B1.

La verificación de los datos de la pestaña *JobData* se deben realizar con trabajos de gran carga de cálculo y de esa forma se observará el estado del *Job* y el reparto de las tareas que se han realizado en los nodos. Esta verificación es correcta y se puede ver en la Figura 14 - Ejecución de trabajos variados. Se verifica también que mientras aparece la ejecución de una tarea en el nodo X que el hardware del nodo X está realizando la tarea.

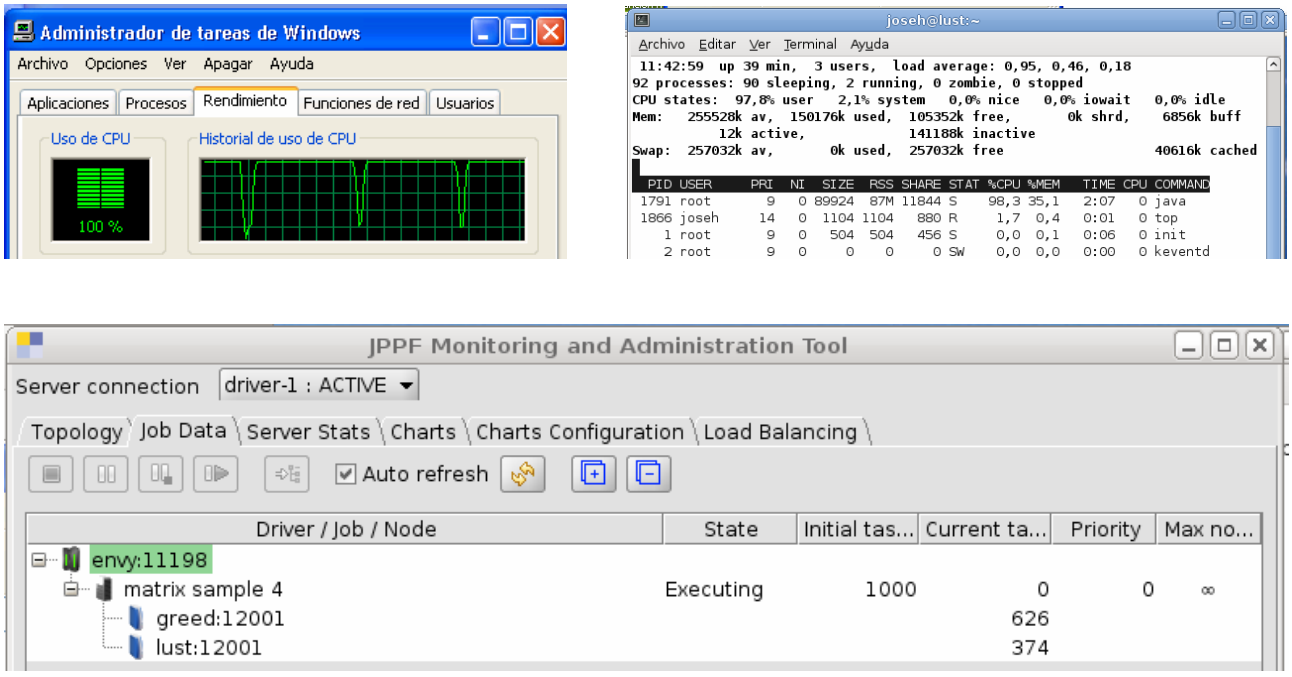


Figura 17 - Comparación de ejecución de procesos en nodos

Ya que en la consola se pueden crear gráficos personalizados sería necesario verificar algunos de ellos y contrastarlos con la ejecución del trabajo que se esté realizando. Se ha utilizado la multiplicación de matrices como medio para introducir cargas en el sistema.

En la primera gráfica de la consola se muestra el proceso de aparición y desaparición de siete nodos. La escala X corresponde con el tiempo y la Y con el número de nodos existentes.

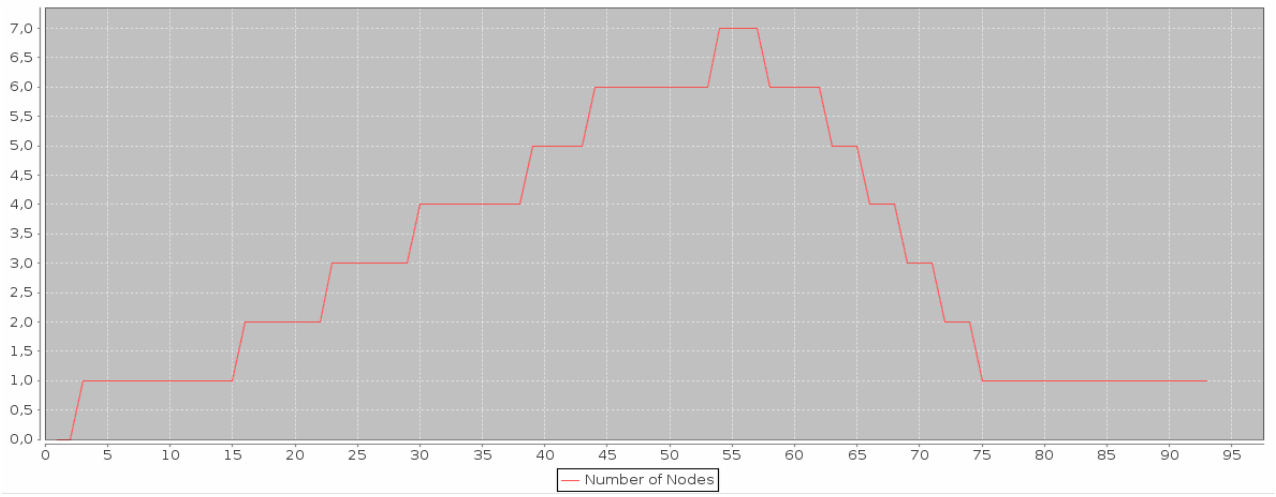


Figura 18 - Inclusión y exclusión de nodos en la infraestructura

En el segundo gráfico mostraremos el número total de procesos que están lanzando cargas al sistema. Se puede observar que todos se ejecutan en paralelo y la finalización es escalonada. Todo ello para una matriz de 1.000x1.000 ejecutada 1 vez por 10 procesos. (Eje X tiempo, eje Y número total de clientes).

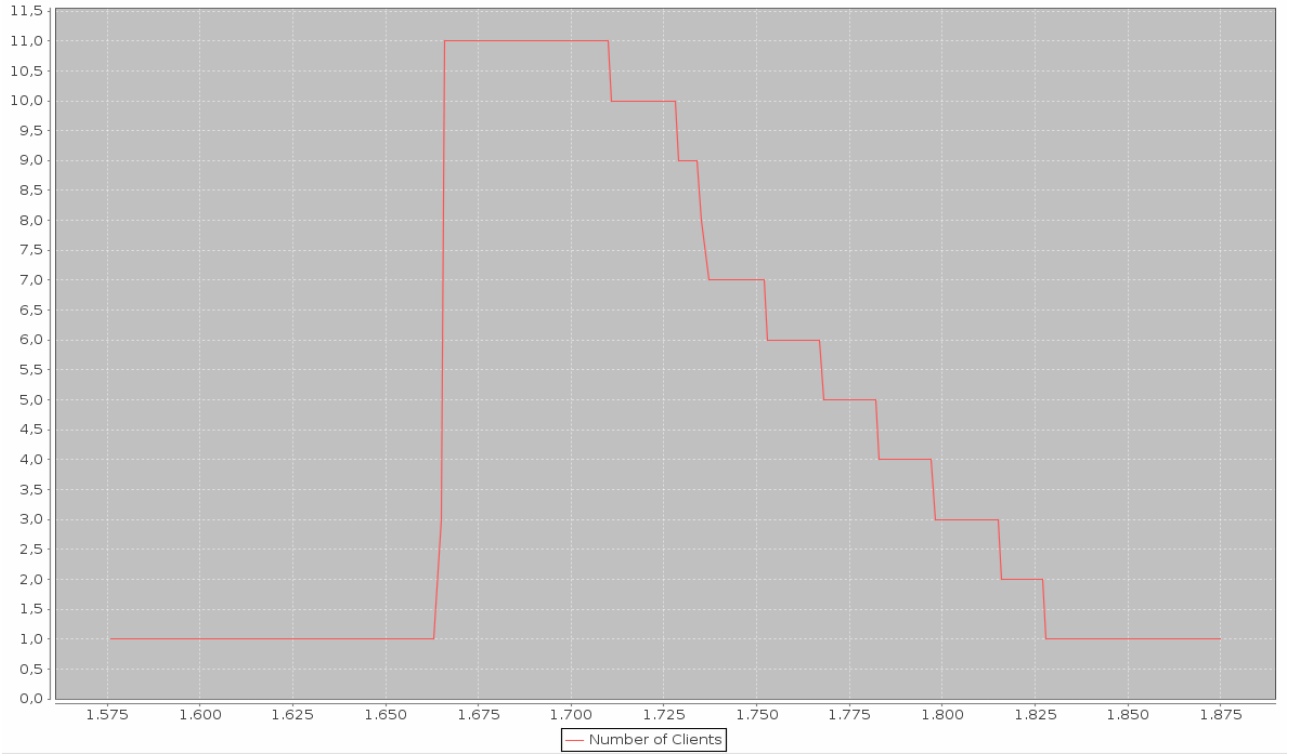


Figura 19 - Procesos con conexión a la infraestructura

El último gráfico muestra el tiempo total de ejecución en los nodos de 10 procesos para una matriz de 1.000x1.000. Como hasta ahora, el eje X corresponde con el tiempo en segundos y el Y con el tiempo de ejecución de los nodos en segundos.

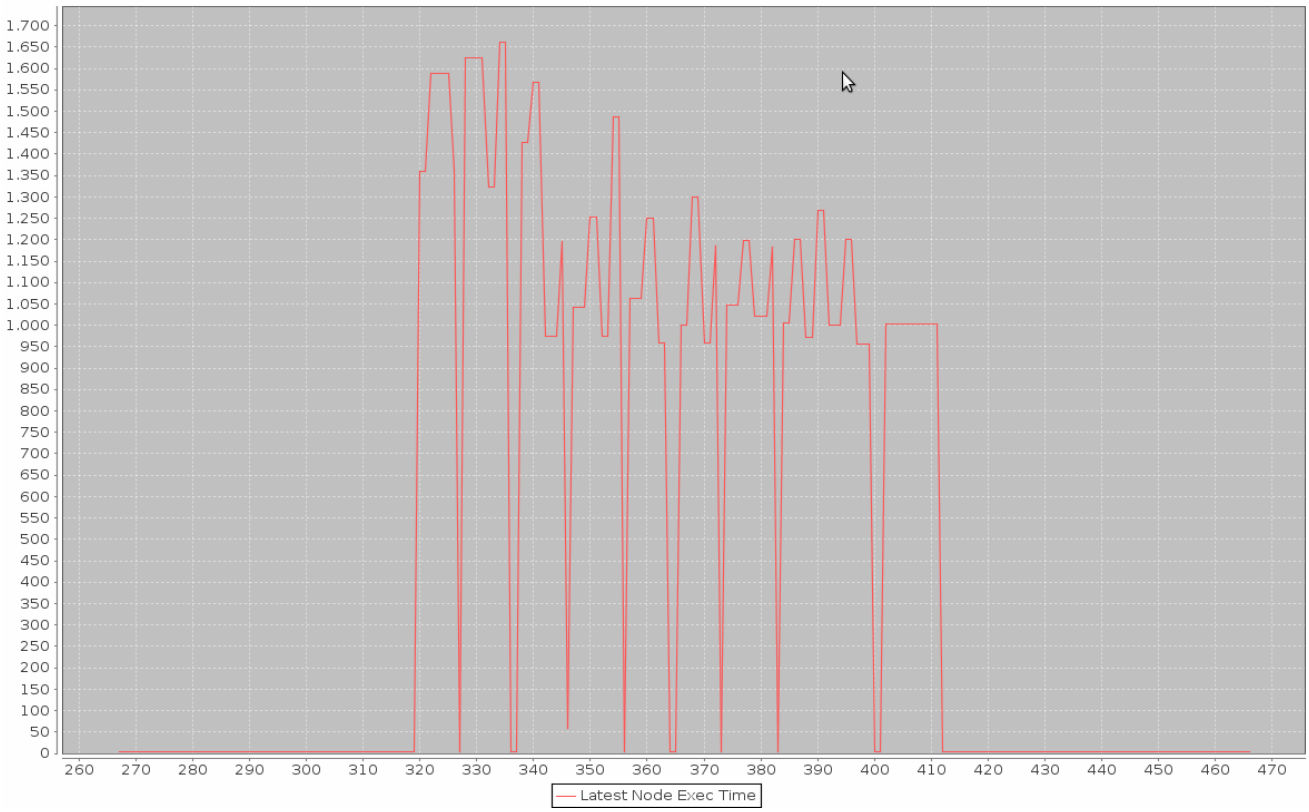


Figura 20 - Tiempo de ejecución en los nodos

Todos los gráficos mostrados son correctos comparados con los datos de ejecución de los procesos realizados.

(E2) Se puede verificar visualmente la actualización de la pantalla durante el proceso de ejecución de los trabajos, por lo tanto no es necesario hacer más pruebas, aunque se echa en falta un lugar, en el entorno gráfico, para cambiar el tiempo de actualización que sí que existe en el fichero de configuración de la aplicación de monitorización.

3.6.- Conclusiones

Como conclusiones de este capítulo podemos realizar las siguientes:

- El framework seleccionado permite una instalación sencilla y tolera la eliminación de nodos de la infraestructura, dependiendo de las necesidades que se posean. La puesta en marcha y parada de Drivers es más compleja, no respondiendo correctamente a las recuperaciones.
- Durante las pruebas se han producido errores por falta de memoria en los diferentes nodos de ejecución, pero esto más que depender del framework es una herencia de las máquinas virtuales de java que se utilizan y de su configuración en el momento de la ejecución de los diferentes componentes utilizados.
- El balanceo de carga es adecuado y permite la utilización de todos los nodos de forma paralela y óptima. No se ha estudiado el sistema de balanceo de carga que utiliza pero la inclusión de nuevos algoritmos para optimización es posible.
- La monitorización de la carga de forma global se ajusta a la realidad y es posible conocer el estado de ejecución de los nodos y el número total de trabajos que se han ejecutado. Se hecha de menos una mayor información sobre el estado de los nodos de forma independiente y la posible evolución de estados o de los tiempos de ejecución sobre los totales de CPU.

- Se puede personalizar para el tratamiento de problemas particulares ya que es altamente adaptable y admite la ejecución de complementos desarrollados, tanto en monitorización como en ejecución.
- No está muy clara la conexión externa del sistema con otros nodos fuera de la LAN propia de la infraestructura. La utilización del multiplexor no es correcta (ver Anexo A4.7).
- Permite tipologías de trabajos desde intensivos en cómputo hasta gran cantidad de pequeños trabajos, con una sobrecarga del sistema asumible.

Parte III - Diseño de estrategias y planteamientos sobre el prototipo a realizar

4.- E-government y gestión de procesos de negocio

Hasta el momento hemos seleccionado y calibrado las capacidades de los entornos grid independientemente de las necesidades de nuestro sistema. Puede llegar a ser un error el evaluar de forma autónoma las herramientas a utilizar sin tener en cuenta el trabajo para el que se seleccionan. Durante este capítulo se mostrarán las características principales de los trabajos que se realizan en los entornos productivos para el gobierno electrónico. Intentaremos organizarlos, describirlos y comentar algunos ejemplos para ilustrar su representación. Es común conocer estos procesos como BPM (*Business Process Management* o Gestión de Procesos de Negocio [10]) aunque tratar en este capítulo estas herramientas en toda su extensión supondría un aumento considerable de los objetivos propuestos.

En general, no se comentará el origen de la información ya que es más bien una mezcla de experiencia junto con la recopilación de algunos documentos obtenidos para ilustrar el funcionamiento y que en ocasiones no son públicos (aunque aquellos que se presentan han sido convenientemente anonimizados).

4.1.- Introducción

Las tecnologías GRID no han sido introducidas en ciertas áreas de la administración y de los negocios para la gestión del día a día. Esto se debe a varios retos pendientes de completar entre los que destacamos:

- Convergencias de los estándares. Es difícil obtener una base común para la representación de los datos y que sea compartida por las distintas partes que componen el problema. Independientemente de su aceptación, debemos tener en cuenta su correcta implantación en los sistemas propios. Como ejemplo podemos indicar JSDL (*Job Submission Description Language*) [05] o incluso la selección de un único framework donde lanzar los trabajos.
- Afianzar y reforzar la seguridad. En ocasiones se plantean problemas de difícil solución para compartir los datos en grandes entornos. Estos problemas no son solo técnicos sino que en ocasiones entramos en el campo legal donde una falta de definición limita muchas implantaciones.
- Definición, despliegue y manejo de servicios en las aplicaciones. No sólo es necesario recibir la información correcta de las diferentes fuentes sino generar la información correcta en función de datos que en ocasiones no están disponibles, son erróneos o carecen de la calidad necesaria.

En este trabajo no se pretende desarrollar un lenguaje de descripción de una aplicación o un motor de flujos de trabajo que sea capaz de trabajar con estructura distribuida. El objetivo es demostrar, que bien a nivel de negocio o en servicios administrativos, se puede crear un entorno en el cual compartir los recursos de forma global y colaborar en la consecución de los objetivos operativos de todos los implicados.

Los **Grids departamentales**, aislados y sin conexión, diseñados para responder a unas necesidades concretas, están extendidos en la investigación y desarrollo, encontrándolos tanto en centros especializados para el tratamiento de información genética humana como en centros de física de partículas. Los **Grids**

empresariales, interconectados para responder mejor a los cambios en la demanda, están implantados, hace tiempo, en entornos de investigación y desarrollo. Las empresas están empezando a ver que con las políticas actuales de reducción de costes, la duplicidad de servicios no es una cuestión deseable. Las **Grids globales** están implantadas en entornos de investigación y desarrollo especialmente en las grandes redes de investigación.

Cuando oímos hablar de Tecnología Grid siempre pensamos en problemas de cálculo extremadamente complejos o bien en algún tipo de producción audiovisual. Sin embargo también se presentan escenarios de orientación comercial, organización interna jerárquica, producción a través de trabajo colaborativo, etc… en el que se han abierto nuevos campos de trabajo y aplicación de esta tecnología. Pero también debemos asociarlas a tareas administrativas.

La realización de tareas dentro de los entornos administrativos viene marcada por la legislación existente. El procedimiento administrativo [16] está regulado por diferentes leyes en los diferentes países y en España por la Ley 30/1992, de procedimiento administrativo [47]. La forma de trasladar los procedimientos al mundo informático es mediante los expedientes electrónicos. Las tareas dentro del mundo administrativo las podemos diferenciar en dos grandes grupos:

- Gestión de Expedientes. Si hablamos de gestión administrativa y trabajo con grandes volúmenes de datos de forma interactiva. Como ejemplo, podemos poner la gestión Tributaria [01] proporcionada por AEAT.
- Gestión de grandes volúmenes de datos. Cuando llegamos al campo de la extracción y resumen de los datos recogidos por sistemas on-line, entrando en el campo de los Data*Warehouse* y el *Business Intelligence* [55].

Esto no quiere decir que dentro de las administraciones públicas no se realicen otro tipo de tareas, pero estas estarán enmarcadas dentro de un expediente administrativo y serán las tareas a realizar dentro del expediente las que tendrán unas características especiales que podrán o no aplicarse la Tecnología GRID.

4.2.- Gestión de expedientes

En esta sección ampliaremos algunos de los escenarios donde se pueden hacer funcionar los entornos grid. La gestión de expedientes es la base principal de la gestión administrativa dentro de las administraciones públicas. Todo trabajo de gestión se basa en la elaboración de un expediente y todas las conclusiones se extraen de un expediente. Las diferentes tipologías de expedientes y su heterogeneidad no permiten diseñar una tarea común para todos ellos, por lo tanto, es necesario definir de una forma clara y ordenada las características y tareas que se realizan en cada uno de los trámites que se gestionan.

En muchas ocasiones la definición de expediente se realiza de forma gráfica mediante un Flujograma o Diagrama de Flujo [54] que permiten entender el trabajo a realizar y su extensión en el tiempo.

Si clasificamos los expedientes dependiendo de su complejidad, podemos hacerlo dentro del grupo de problemas intensivos de datos y en ciertos casos pasan a ser intensivo en datos y procesos. Para ello hemos tenido en cuenta el gran volumen de información que se utiliza y, en ciertas ocasiones, la complejidad de la realización de procesos de extracción de datos que en un principio no estaban contemplados.

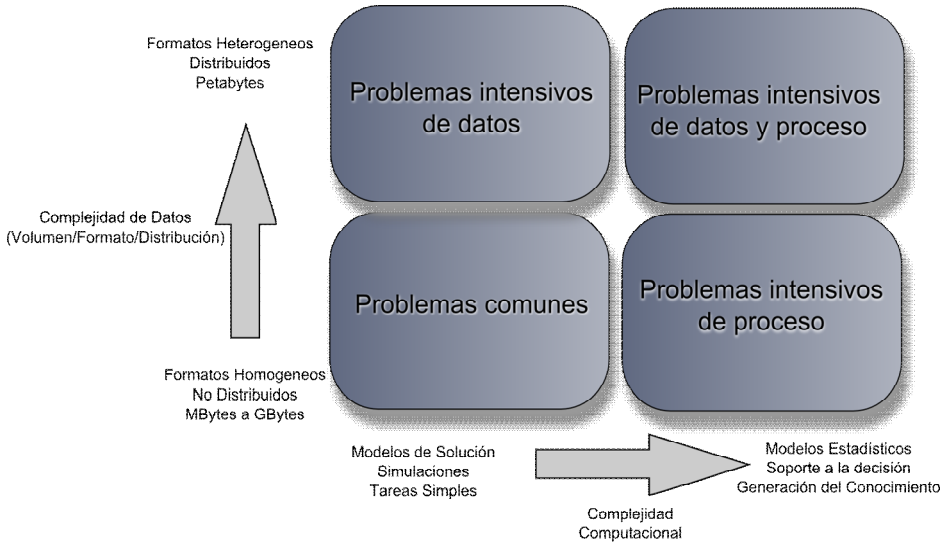


Figura 21 - Clasificación de los procesos. De [32]

4.2.1.- Representación del procedimiento

Históricamente, las aplicaciones informáticas que dan soporte a la gestión de expedientes en las administraciones públicas se han desarrollado y evolucionado como soluciones independientes para cada sección o tipo de gestión. Las tareas, que poseen diferente duración, requisitos variados, circuitos administrativos complejos y documentación extensa, resuelven su problema con soluciones dispares y de costosa evolución.

Existen numerosas propuestas que, en general, pretenden solucionar este problema. Se plantean soluciones integrales y, en general, cada una de las administraciones las solventa de una forma diferente tanto estructural como tecnológicamente.

Es importante reconocer que el objetivo de este trabajo no es realizar una solución entregable para la gestión integral de expedientes en las administraciones públicas, ya que existen suficientes y de gran calidad. Siempre que se ha planteado este trabajo se ha centrado en verificar la viabilidad del cambio de paradigma con el estudio de los tratamientos de información en las administraciones públicas. No es tan importante en estos momentos el realizar un producto con todas las funcionalidades desarrolladas, como el verificar que el tratamiento de expedientes mediante la tecnología GRID puede proporcionar más beneficios que cualquier otra propuesta de trabajo que en la actualidad se realice.

Lo primero que se ha realizado para poder tratar los expedientes es verificar el método para el diseño de los mismos. Los modelos de representación de los expedientes son variados utilizando los diagramas de flujo de gestión de trabajos (ver Anexo 7).

4.2.2.- Modelos de representación utilizados en este proyecto

La finalidad de este trabajo no es realizar una enumeración de los posibles trabajos existentes en la administración pública. Sería demasiado extenso. Pero sí que se puede facilitar la comprensión de la adaptación a JPPF que se pretende mediante la presentación de un ejemplo simplificado de un procedimiento administrativo. Este circuito de pruebas cubrirá numerosos ejemplos de trámites a realizar así como el procedimiento de inicio y fin.

En la siguiente imagen se describe la representación utilizada:

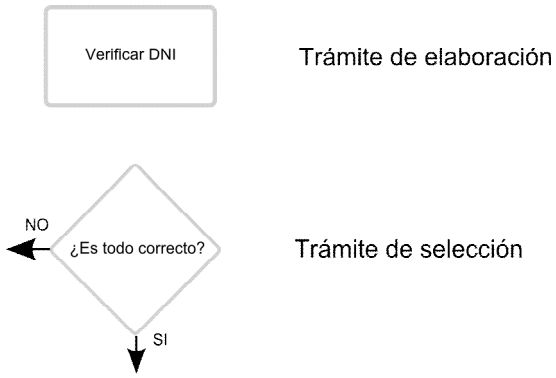


Figura 22 - Representación de los procedimientos

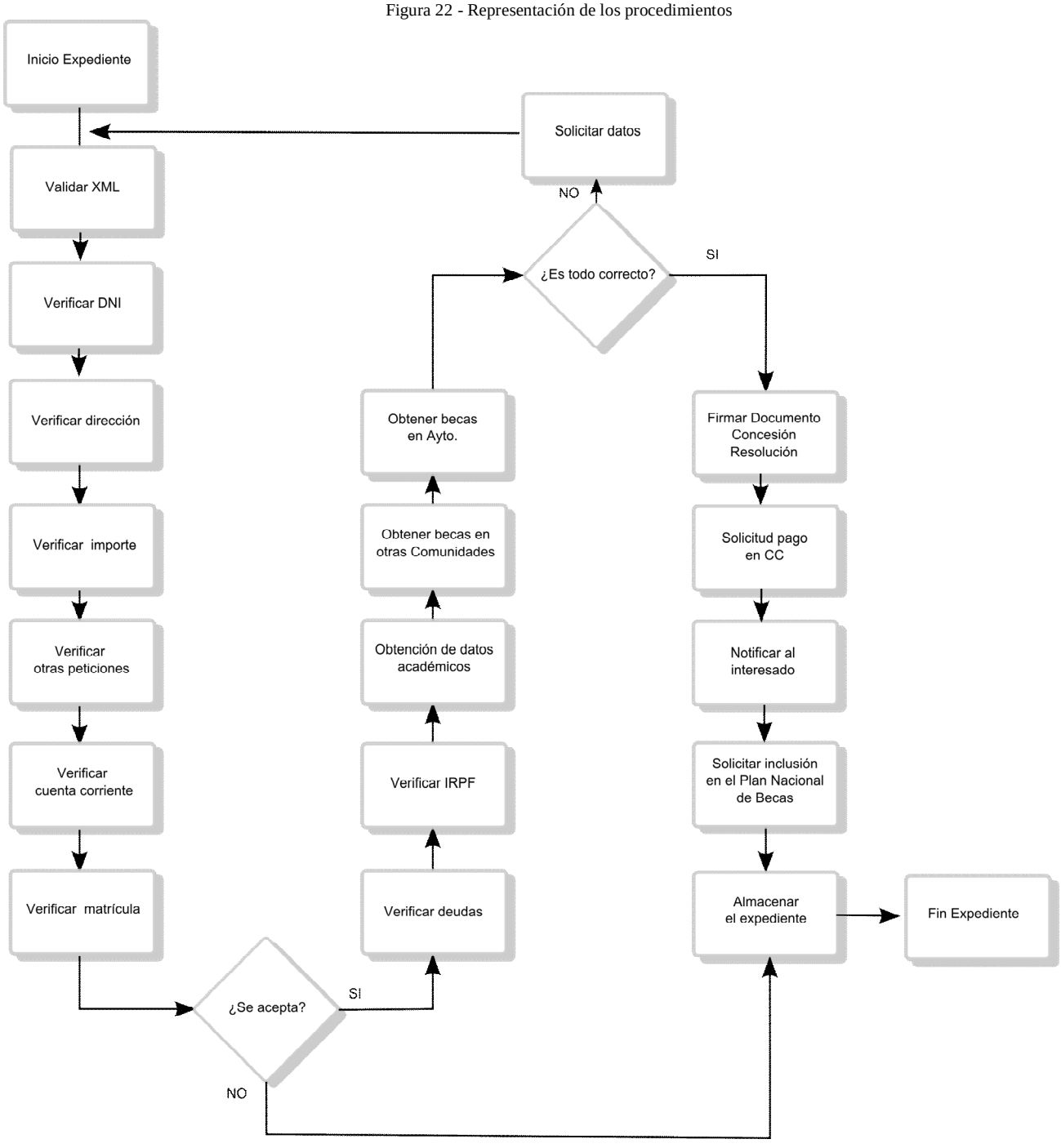


Figura 23 - Flujograma de ejemplo tratamiento de beca

El tratamiento completo recibe el nombre de procedimiento administrativo y cada una de las fases en el que se ha dividido hasta llegar a tareas realizables recibe el nombre de trámite administrativo.

Supongamos un expediente como el que se muestra en la Figura 23 - Flujograma de ejemplo tratamiento de beca y que corresponde a una tramitación simplificada de una beca. Esto nos servirá como base para llevar a cabo las pruebas. Como podemos observar, la totalidad de la gestión se ha realizado de una forma secuencial, sólo modificado por dos decisiones. La representación se ha simplificado en gran medida para adaptarse a las necesidades del ejemplo que se quiere realizar. La utilización de un ejemplo real no aclararía mucho la explicación que se pretende.

Con este circuito se ha reducido el problema a una secuencia de tareas a realizar de forma local o remota de las cuales obtenemos un resultado deseado y el almacenamiento del expediente.

Una breve descripción de las tareas a realizadas es la siguiente:

- Inicio del Expediente. En este trámite se incluyen todos los procedimientos previos a la introducción del expediente dentro del grid.
- Validar XML. El expediente se representa como un XML (Anexo 1) y se valida contra un XSD (Anexo 2).
- Verificación de NIF's. Se realizará una verificación de los números. En esta fase se lleva a cabo todas las verificaciones de NIF's que existan en los expedientes. Lo más lógico sería que esta labor se realizara en el organismo que tiene competencia para la asignación de los números.
- Verificar Dirección. Se comprobará la dirección que se posee contra un servicio de callejero o bien contra el NIF anteriormente indicado. Supongamos para simplificar que se realiza contra algún sistema con información catastral.
- Verificar Importe. Comprobación que el importe de la solicitud está en el rango adecuado del tipo de beca solicitada.
- Verificar otras peticiones: Comprobar que no se han realizado otras peticiones de beca en el sistema.
- ¿Se acepta? El tramitador del expediente verificará que todos los datos son correctos.
- Verificar la cuenta corriente. Realizar la comprobación que la cuenta corriente corresponde con el NIF proporcionado como titular o autorizado. Esta información solo la poseen los bancos.
- Verificar la matrícula. Se comprueba la existencia de una matrícula para el NIF indicado en el centro indicado y el curso indicado.
- Verificar deudas. En el caso de tener que realizarse algún pago se verifica que no existen deudas pendientes para los NIF participantes.
- Verificar IRPF. Verificar que la declaración de la renta está en el rango adecuado de las casillas necesarias. Sólo la AEAT puede proporcionar esta información.
- Obtención de los datos académicos que aseguran que puede solicitar la beca.
- Obtener las becas solicitadas, en otras comunidades autónomas o a nivel nacional.
- Obtener las becas solicitadas en el ayuntamiento del centro solicitado.
- Firmar el documento. Corresponderá con la generación del documento electrónico que permite el pago. Esta resolución será firmada electrónicamente por el responsable y añadida al expediente.
- Solicitar el pago de una cantidad a una cuenta determinada por parte del organismo competente. Es común que sea otro departamento el que realice el pago.
- Notificar al interesado. Informar vía mail/carta de los detalles concedidos y las fechas previstas de pago.
- Solicitar la beca nacional que cubre cierta parte del pago realizado según el concierto realizado.
- Guardar el expediente. Archivar el expediente electrónico en un lugar para su almacenamiento.
- Solicitar datos. El sistema debe enviar un mail solicitando documentación adicional y después esperar a que se introduzca en el sistema.
- Fin del expediente. Trámite en el que se da por terminado.

Podemos observar que los trámites, en su mayoría son verificaciones de los detalles de los documentos presentados. En algunas ocasiones estas verificaciones deberían realizarse en sistemas externos a los que dispone el sistema de gestión ya que la información pertenece a otras entidades públicas.

Por el momento únicamente hemos diseñado el tipo de trabajo a realizar y las fases que posee. Se ha representado todo ello mediante un diagrama de flujo. Para poderlo ejecutar en diferentes ubicaciones deberemos indicar de alguna forma el lugar en el que se puede ejecutar. Este sistema de representación no aclara la localización. La labor de localización se realizará mediante otra representación.

4.2.2.- Introducción de los lugares de decisión y ejecución

En el punto anterior ya hemos descrito brevemente el lugar donde se deben ejecutar cada uno de los trámites. En ocasiones este trámite se puede realizar de forma local y en otras se realiza de forma remota o en una localización exterior al sistema.

Es necesario definir previamente el lugar de ejecución para cada una de las tareas que se desean realizar._Para ello nos ayudarán los gráficos de los procedimientos administrativos que se exponen a continuación.

4.2.2.1.- Representación gráfica de los procedimientos administrativos

Para la representación gráfica de los procedimientos administrativos se pueden utilizar diferentes modelos de representación. A continuación se describe el utilizado por el Sistema SUR de Gestión Tributaria en Andalucía. Estos diagramas pueden contener los siguientes símbolos:

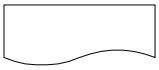
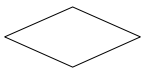
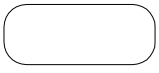
Nombre	Descripción	Representación
Trámite	Conjunto de funcionalidades que agrupan un paso en la gestión de un expediente.	
Documento	Simboliza documentos que presenta el contribuyente o que se le envían a éste.	
Decisión	Este símbolo se utiliza para elegir entre dos caminos en el circuito.	
Conector entre páginas		
Procesos del sistema	Funciones que se realizan de forma automática y que son transparentes al usuario.	

Tabla 6 - Símbolos para la representación

En la siguiente figura se muestra un ejemplo de esta representación:

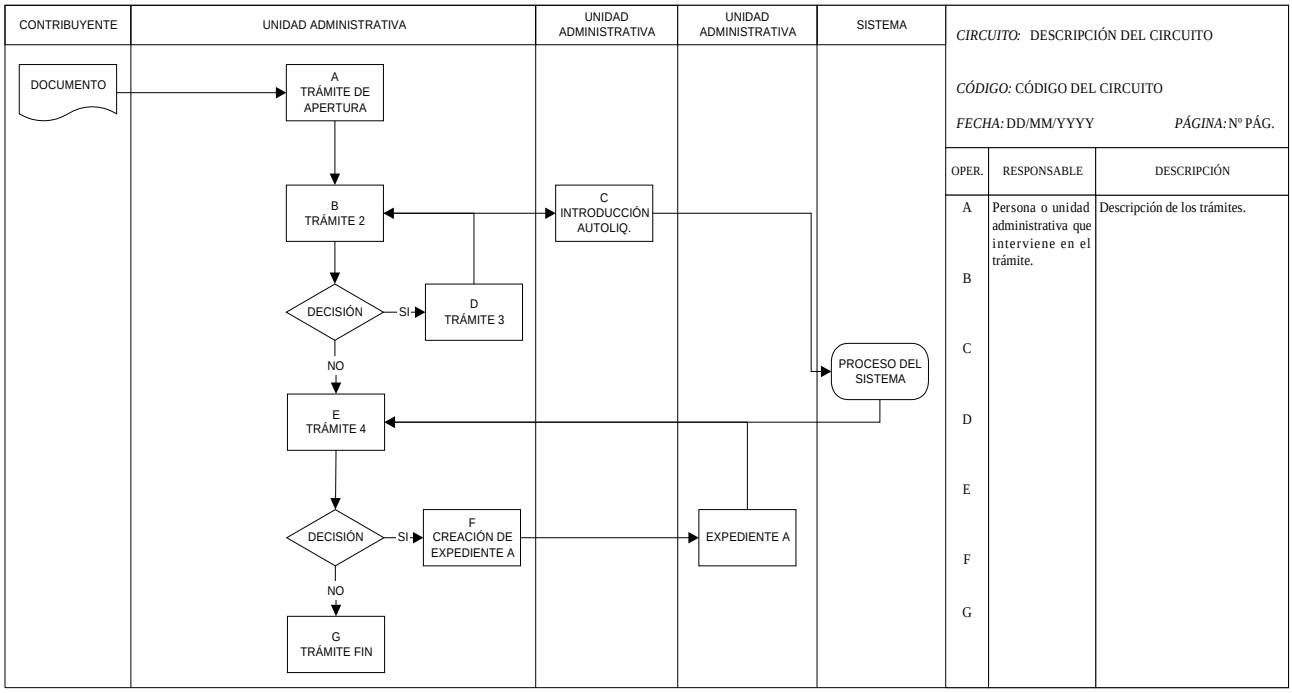


Figura 24 - Representación gráfica del procedimiento administrativo

En este esquema base se ha descrito un trámite genérico que tiene como origen el contribuyente. En un principio, casi toda la tramitación la realiza la primera unidad administrativa. La introducción de la autoliquidación la realiza una unidad administrativa diferente y la generación del expediente A, otra unidad distinta. Como vemos, es sencilla la asimilación de los lugares de ejecución a los nodos de trabajo teniendo en cuenta las columnas donde se sitúa.

4.3.- Extracción de grandes volúmenes de datos

Otros procesos que tienen que ver con la información que se les proporciona a los usuarios finales serían las extracciones de datos mediante los procesos batch o desasistidos. Sobre esta cuestión podríamos hablar de diferentes tipos de procesos de generación de informes, pero en la actualidad la generación de datos de forma agregada tiene una tecnología asociada que puede facilitar mucho la tarea de extracción y representación de los datos. La inteligencia de Negocio o BI (*Business Intelligence)* permite realizar una explotación en los ambientes donde el gran volumen de datos imposibilita su tratamiento on-line.

Entre los procesos que más importancia tienen están las ETL (Extracción, Transformación y Carga de datos).

4.3.1.- ETL (Extract, Transformation and Load)

No todos los trabajos que se realizan en las administraciones públicas corresponden con el tipo de expediente que se ha descrito hasta el momento. Algunos trabajos se realizan, bajo una visión de trabajo con gran volumen de datos y pocas tareas. Este tipo de trabajo entra en contraposición con la gran cantidad de tareas y datos limitados que se tratan en la gestión de expedientes.

Los procesos de Extracción, Transformación y Carga (ETL – *Extract, Transform and Load*) son propios de grandes organizaciones y permiten mover datos de múltiples fuentes, reformatearlos, limpiarlos y cargarlos en bases de datos, *DataMarts* [09] o *Datawarehouses* [14] con el objetivo de apoyar el proceso de negocio o bien para la integración de sistemas heredados.

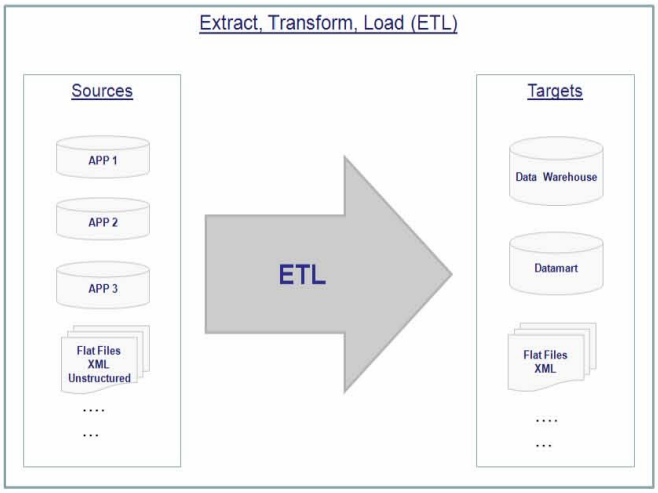


Figura 25 - Representación de la Extracción, Transformación y Carga

La primera fase del proceso, la extracción se refiere a la selección de la fuente de los datos y los procesos para su extracción. Es bastante común poder realizar procesos de extracción en diferentes bases de datos con diferentes formatos de datos. En esta fase son comunes las Bases de Datos Relacionales pero también son utilizados los ficheros planos y cualquier sistema de generación de información de las organizaciones implicadas. También está dentro de esta primera fase la verificación de la corrección de los datos y su acondicionamiento.

En la segunda fase, el proceso de transformación, podemos encontrar numerosas tareas como el cambio de codificación, la obtención de nuevos valores, la ordenación, la transposición de filas y columnas y la aplicación de validaciones complejas.

En la tercera fase, de carga, se agregarán los datos a un *Datawarehouse* dependiendo de las necesidades de la organización. Puede guardarse la información ya agregada o bien historizada. La forma de sobrescribir los datos o añadir los nuevos es una estrategia a tener en cuenta en esta fase.

Los procedimientos de paralelización en la extracción, transformación y carga de datos se pueden diseñar de la misma forma que se ha realizado el tratamiento de los expedientes aunque, por el tipo de trabajo que se realiza, existirán muchas más tareas paralelas y muchos más datos que tratar en cada una de ellas.

Ya podemos ver que el diseño de este tratamiento se parece mucho al anterior proceso de los expedientes, en la forma de representación, y que podemos realizar un diseño de ETL con las herramientas planteadas para ello seguiremos este diseño junto con el proceso de diseño de expedientes, aunque debemos tener en cuenta que la fuente de datos de uno y otro caso es de diferente volumen.

4.4.- Taxonomía de las tareas

Se ha explicado la necesidad de tener un procedimiento claramente definido, que puede representarse gráficamente. Éste, internamente, se compone de numerosas tareas a realizar. En esta sección intentaremos profundizar en sus características y se organizarán de una forma racional y práctica en base a la experiencia.

Existen numerosas propuestas para la gestión electrónica. En [17] se expone la diferenciación entre 3 partes:

- Componentes normalizados. Que son los encargados de realizar las funciones más complejas de combinación, interacción e integración.
- Relaciones preestablecidas. Que son los vínculos existentes entre los componentes que aplicándolos de diferente forma dan lugar a múltiples diseños.
- Principios: Conjunto de reglas que dan coherencia y consistencia a las relaciones entre componentes.

Para el caso que nos aplica diseñaremos los componentes por medios de Job independientes y las relaciones y principios los modelaremos mediante una secuencia general a ejecutar.

En este apartado se propone una taxonomía de los componentes para incluir los tipos de trabajo presentes, poniendo el punto de vista en la tecnología GRID.

- Funciones remotas. Serán aquellos procesos que se solicitan a un nodo remoto con el fin de obtener una transformación de los datos remitidos. En general producen una validación y/o modificación de los datos ya existentes en el sistema. Estas funciones no necesitan intercambio de información fuera de los límites de la infraestructura. Las podemos clasificar en función de su complejidad:
  - De Consultas. Podemos entender como consultas aquellas funciones en que se solicita la validación de un dato remitido como argumento o bien se desea obtener un dato no estructurado generado por otro sistema o aplicación.
  - De Generación/Almacenamiento de datos. Este tipo de función tiene como fuente o resultado datos complejos. El objetivo es la creación o recolección de información elaborada.
- Tareas de Entrada/Salida de información. Estarán dentro de este grupo aquellas invocaciones a nodos remotos para que ejecuten algún tipo de programa que permita la entrada/salida de información bien vía un fichero o bien de otro sistema.
  - Interactivos. Son aquellos que obligan a actuar a algún usuario/proceso en el mismo instante.
  - No Interactivos. Estarían en este grupo los trabajos desasistidos que no necesitan actuación de un usuario/proceso en el mismo instante.

Si tenemos en cuenta los trabajos en función de las situaciones posteriores a su ejecución deberemos clasificarlos en:

- Con Espera. Una vez introducido el trabajo en el sistema se debe tener la posibilidad de almacenar la petición en algún lugar para indicar el estado en el que se encuentra. Esta capacidad se debe extender a la posibilidad de poder reiniciar la tarea de nuevo. Los trabajos en espera siempre estarán en este estado en función de la terminación de otro trabajo o bien en función del tiempo.
- Sin espera. Estos trabajos no deben esperar la finalización correcta de ningún trabajo lanzado con anterioridad. Por ello pueden continuar su ejecución de forma normal.

Si cruzamos las dos clasificaciones vemos el siguiente cuadro:

		Sin Espera	Con Espera
Tareas de E/S	NO interactivos	SI (A)	SI (E)
	Interactivos	SI (B)	SI (F)
Funciones Remotas	De Consulta	SI (C)	N/A (G)
	De Generación de datos	SI (D)	N/A (H)

Figura 26 - Taxonomía de trabajos GRID

Analizando la tabla realizada podemos ver que las funciones remotas, tanto de consulta como de generación de información no tienen sentido que se realicen con espera (G y H) ya que si no se dispone de los datos es porque es necesaria una introducción exterior de los mismos y por lo tanto serían E y F. Las funciones C y D deben gestionarse sin espera o una espera razonable.

Las tareas de E/S tienden a ser más complejas ya que se pueden gestionar con espera o sin espera. En el caso de NO interactivos con espera (E) se refiere a tratamientos de datos que necesitan la validación o bien la activación por parte de un tercero pero no en el mismo instante. Los interactivos con espera (F) son aquellos que se debe esperar a que se realicen por una persona pero que no tienen un tiempo definido. Los trabajos NO interactivos sin espera (A) se refieren a la búsqueda de un fichero o variables en un lugar de

almacenamiento. El tiempo en espera debe ser mínimo. Los interactivos y sin espera (B) son aquellos que necesitan, en ese mismo instante, la actuación de una persona para finalizar correctamente.

Con el fin de ilustrar la clasificación pongamos unos ejemplos sencillos de cada uno de los tipos:

- Tipo (A). Se envía por e-mail un documento de notificación o aviso.
- Tipo (B). El gestor indica que el expediente puede seguir el siguiente trámite.
- Tipo (C). Comprobar la veracidad de la firma electrónica de un documento.
- Tipo (D). Obtener un sellado de tiempo para un documento. Obtener registro de títulos para un NIF. Certificado de estar al corriente de los pagos a la Seguridad Social, etc.
- Tipo (E). Se espera que se presente un documento solicitado.
- Tipo (F). El proceso debe esperar a que se firme digitalmente por un responsable alguno de los documentos que compone el expediente.
- Tipo (G). NO es aplicable.
- Tipo (H). NO es aplicable.

4.5.- Expediente electrónico

No quedaría completo este capítulo si no dedicáramos algunas líneas a las propuestas de estandarización de los expedientes. Existen numerosas propuestas para la estandarización de los expedientes electrónicos como puede ser la propuesta latinoamericana que se hace en Uruguay [20]. En España uno de los puntos más desarrollados técnica y jurídicamente es la digitalización certificada de los documentos.

Casi todas las propuestas que se han podido estudiar tratan de almacenar la información de un expediente en un fichero XML con más o menos complejidad. Este expediente posee información sobre los ficheros que lo componen e información adicional sobre los diferentes procesos que están realizando.

En [21] podemos encontrar una propuesta de expediente en formato electrónico realizado por el Ministerio de Política Territorial y Administración Pública en el marco del "*Esquema Nacional de Interoperatibilidad*".

Para nuestra propuesta de trabajo debemos encontrar un medio de almacenamiento de estos expedientes. Por un lado, tenemos las diferentes propuestas realizadas que resultan complejas y laboriosas de implementar, por otro lado, se puede realizar una propuesta simplificada que pueda admitir los numerosos tipos de expedientes.

Se ha optado por la segunda solución. Mediante un proceso de recopilación y simplificación de los ejemplos anteriormente expuestos se ha planteado una propuesta que sea utilizable para las pruebas. Para nuestro caso, un expediente electrónico será un fichero con una pequeña cabecera de información del expediente y una parte con los ficheros que componen la carpeta. Un ejemplo de documento se puede ver en el Anexo 1 con su XSD de validación en el Anexo 2.

5 .- Propuesta de las estrategias de trabajo

En este capítulo se detallarán todos los aspectos para la realización una propuesta completa y verificable en el tratamiento de los flujos de trabajo por medio de la tecnología seleccionada. En los capítulos anteriores se ha mantenido independizado el diseño estructural de la funcionalidad. De ahora en adelante, intentaremos juntar los modelos de diseño con las propuestas de utilización de la herramienta, de esta forma, se mostrarán soluciones de las distintas problemáticas que aparecen en la realización del proyecto.

5.1.- Propuesta de arquitectura

La propuesta de arquitectura se basa en la utilización de un grid para el tratamiento de los trabajos introducidos por los diferentes medios habilitados. Se han dispuesto cuatro capas con la siguiente descripción:

- Capa Cliente – También conocida por capa de presentación o interface. La capa cliente se compone de las máquinas de los usuarios finales junto a sus navegadores que realizan peticiones a los servidores Web.
- Capa Web – Esta capa se basará en la existencia de varios servidores web que permitirían responder a las peticiones realizadas por la capa cliente.
- Capa GRID – Soporta la realización de las tareas encomendadas, realizando la función de balanceo de carga entre los diferentes nodos que existen en el sistema.
- Capa Datos – Se compone de las estructuras de almacenamiento que se diseñen. Pueden ser bases de datos de gran tamaño o incluso sistemas de ficheros compartidos.

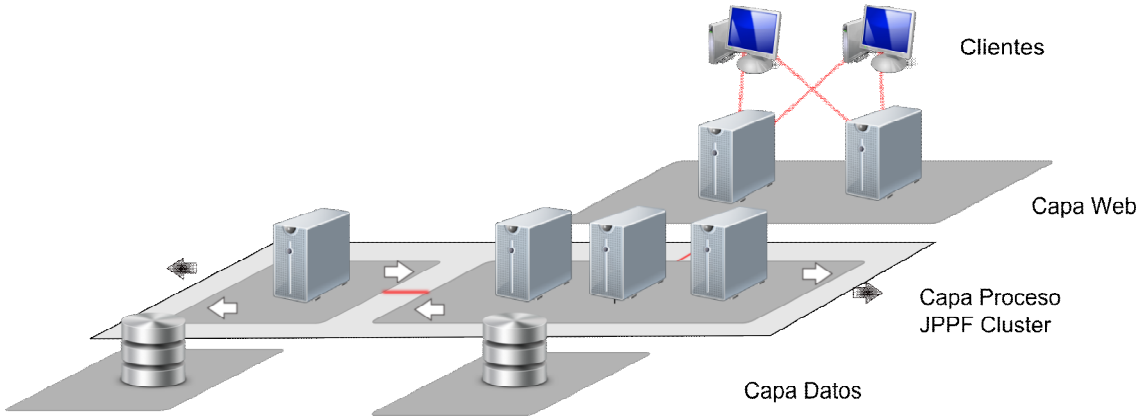


Figura 27 - Arquitectura multicapa GRID

Las peticiones llegarían por medio de los servidores web que sirven al sistema como interfaz de acceso, junto con la capa de presentación que son los navegadores en los ordenadores de los usuarios finales.

Una vez introducido el expediente dentro del sistema, éste se ejecutará dependiendo de sus trámites administrativos en los servicios del sistema. Un tipo de servicios puede estar dimensionado con n nodos para el correcto funcionamiento del entorno y de la carga a realizar. Cada nodo podrá acceder a una base de datos local o bien en el caso de ser un nodo remoto a su base de datos o estructura de datos remota.

Los trámites administrativos podrán ejecutarse de forma paralela en diferentes nodos sin tener que preocuparse de la situación de la ejecución de cada uno de ellos. Los trámites de los diferentes expedientes se podrán ejecutar simultáneamente.

Una vez terminada la tramitación o si es necesario algún tipo de comunicación con el exterior durante el proceso, se notificará a la capa web que permitirá el intercambio de información con el usuario.

5.2.- Servicios básicos para la infraestructura

Adelantándonos a la infraestructura y a la filosofía de los trabajos tratados como tareas dentro de un sistema grid, parece evidente poder implementar los siguientes procesos básicos para el correcto funcionamiento del sistema:

- LogBook. Mantendrá una bitácora centralizada de los mensajes que se producen durante la ejecución de los expedientes.
- Record Broker Manager (RBM). Se encargará de introducir los expedientes y de mantener en ejecución un número de expedientes adecuado dentro del sistema.
- Almacén de documentos. Recibirá los ficheros y mantendrá un sistema de localización de los mismos dentro del sistema.

5.2.1.- Desarrollo jerárquico en la nomenclatura de las tareas a realizar

Ya que la ejecución de los trabajos debe realizarse en lugares concretos, en este punto entraremos en la denominación de los mismos y la forma de diferenciarlos. Aunque las herramientas no obligan a utilizar una estructura jerárquica en la nomenclatura de los servicios de los que disponen los nodos, se ha visto útil realizar una estructura jerárquica que recoja todos los posibles servicios existentes.

Esta estructura no condiciona la arquitectura física o lógica del sistema. Lo único que marca es una denominación de las tareas para poderlas organizar. La utilización de esta estructura es aconsejable pero no necesaria para que el sistema funcione correctamente.

En la estructura jerárquica los servicios se denominarán de la siguiente forma:

(tipo de nodo).(nombre del servicio).[tipo de servicio]

O bien

general.(nombre del servicio).[tipo de servicio]

Como ejemplos podríamos indicar:

ayto1.becas_ayto *ga1.se_acepta.solicita* *ga1.se_acepta.responde*

o como común

general.logbook

5.3.- Tratamiento de los diferentes tipos de trabajo

En función de la clasificación de tareas, definida en el capítulo anterior, podemos realizar las siguientes propuestas de implementación:

Tipo (A) – Funciones de E/S no interactivas sin espera. (*Como ejemplo podemos utilizar la notificación a un tercero*).

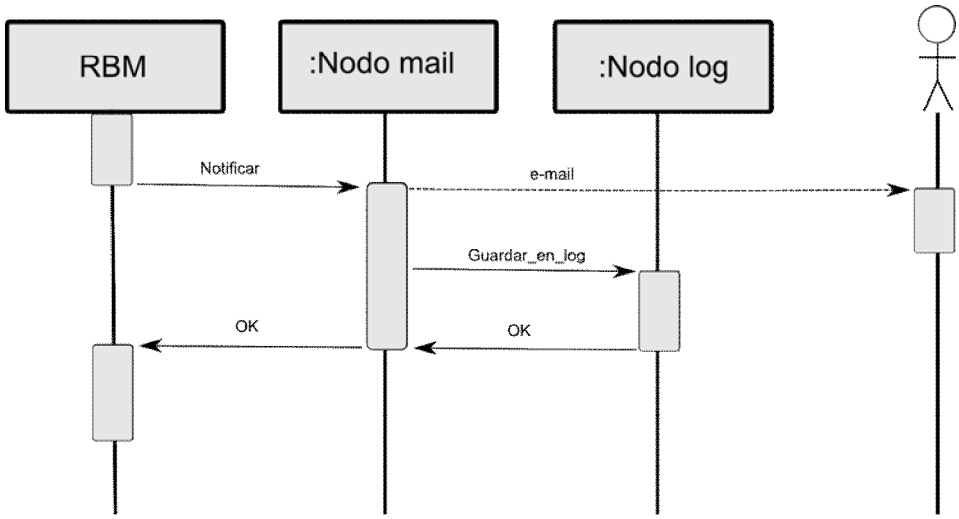


Figura 28 - Propuesta de tratamiento de tareas tipo A

Como vemos en el esquema, el RBM o un nodo de tratamiento general realiza una petición a un nodo dedicado al envío de correos electrónicos. Este se encargará de realizar el envío por medio de la infraestructura que sea necesaria para que llegue al usuario final. Al terminar el trabajo en el nodo se puede grabar la acción en un nodo dedicado a guardar la información de la traza de los trabajos realizados. Todo esto retorna una simple comunicación de finalización de la operación correcta o incorrecta.

Tipo (B) – Funciones de E/S interactivas con espera (*Como ejemplo podemos utilizar la validación de un gestor que se encarga de indicar al sistema que el procedimiento puede continuar*).

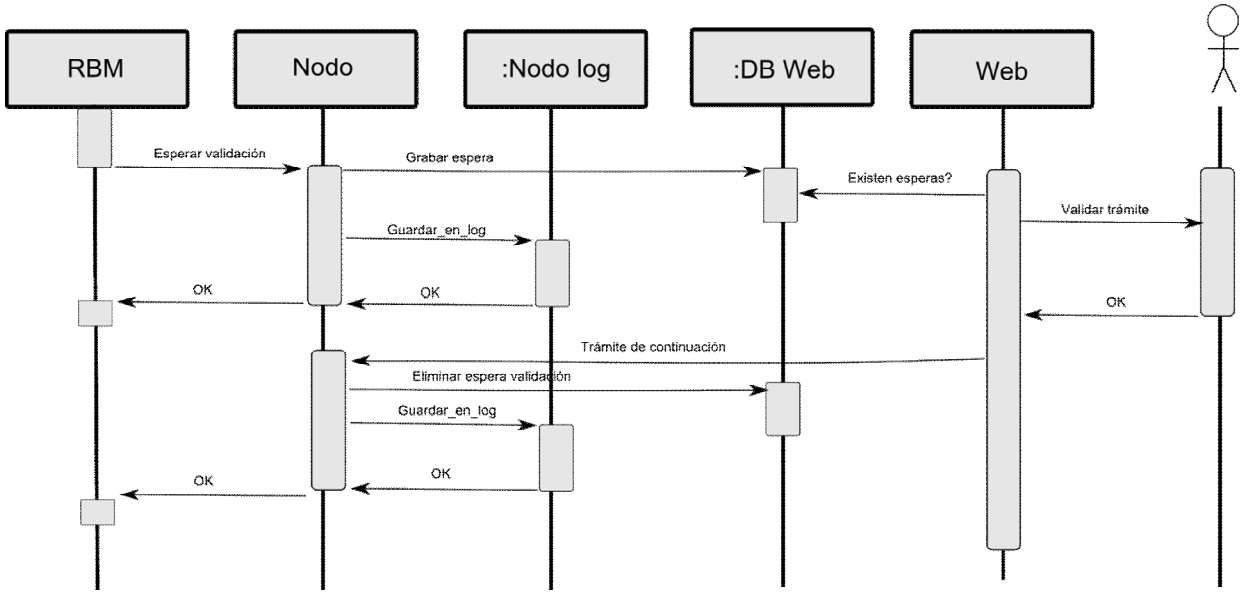


Figura 29 - Propuesta de tratamiento de tareas tipo B

En este caso el RBM o nodo que está ejecutando el expediente es el encargado de remitir a un nodo de tratamiento general de expedientes la realización de una espera de tipo B. Este se encarga de guardar la información de espera necesaria, que posteriormente será accedida por la capa web y se la presentará al usuario final. Este la aceptará y creará un nuevo trabajo que le comunicará al RBM que ya puede continuar con la tramitación.

Tipo (C) - Funciones remotas de consultas sin espera. (*Como ejemplo podemos utilizar el verificar la firma electrónica de un documento*). En este caso la ejecución de los trabajos se realizará exclusivamente en un nodo con las capacidades necesarias.

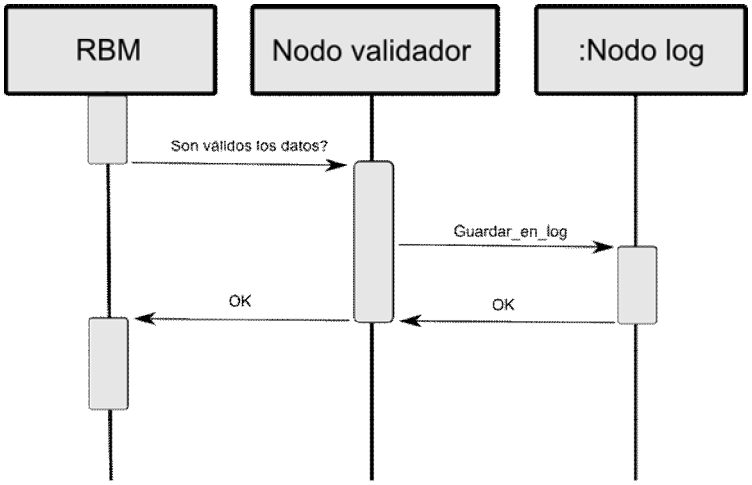


Figura 30 - Propuesta de tratamiento de tareas tipo C

Tal vez sea uno de los ejemplos más sencillos y que más puede beneficiar a la infraestructura. Es el nodo validador el que posee toda la información para la verificación de una firma electrónica, es por ello necesario enviar el documento y recibir la respuesta de si es o no válido. También se hace la grabación en el log.

Tipo (D) – Funciones remotas de generación de datos sin espera. (*Por ejemplo, obtener el sellado de tiempo para un documento*). A la implementación de estos trabajos se les aplicará los mismos principios que los de tipo A aunque no se envía a una persona sino que se guarda un documento en el Almacén de Documentos.

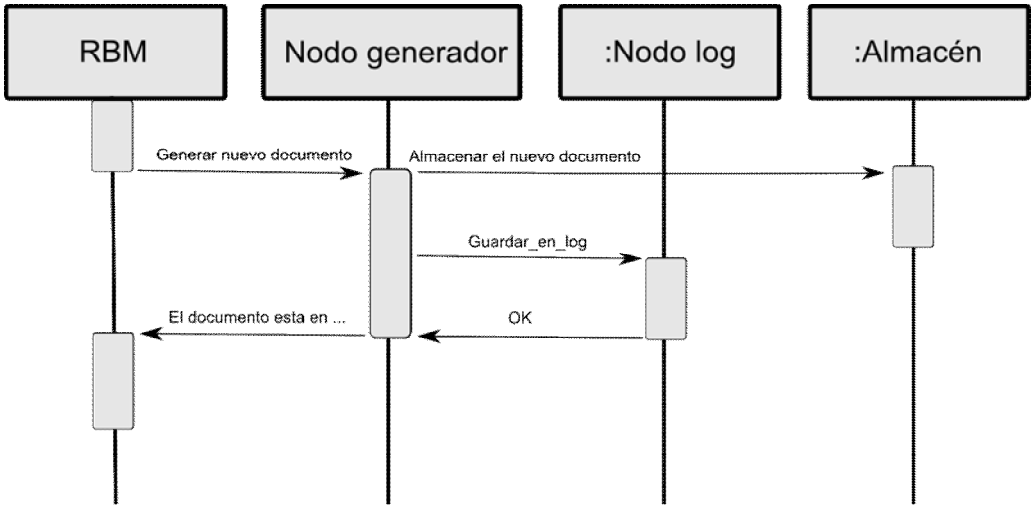


Figura 31 - Propuesta de tratamiento de tareas tipo D

El Almacén de Documentos puede ser cualquier infraestructura que pueda guardar los ficheros. Podríamos hablar de un sistema de archivos, un FTP que tal vez en nuestro caso sería más eficiente con GridFTP [33] o bien un servicio de gestor documental estilo Alfresco [02], que da numerosos servicios para el almacenamiento de ficheros. Tengamos en cuenta que el servicio de almacén es la conjunción de la referencia a la localización y el lugar donde se encuentra.

Tipo (E) – Funciones de E/S no interactivas con espera (*Por ejemplo esperar a la presentación de un documento solicitado*).

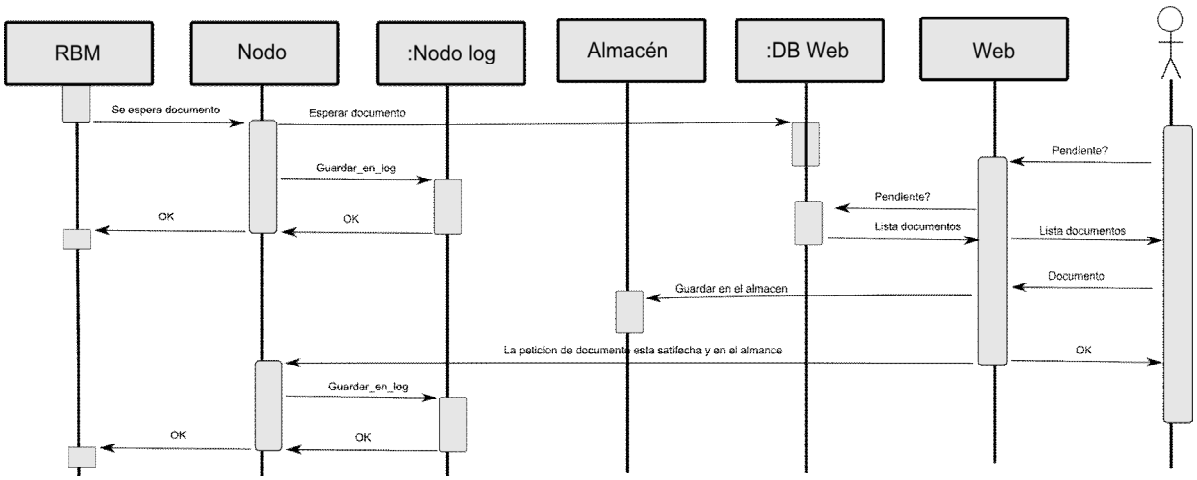


Figura 32 - Propuesta de tratamiento de tareas tipo E

El procedimiento comienza con la grabación en la base de datos web de la información de los documentos que es necesario presentar. Así, cuando llega el documento se le asocia a esta petición y se lanza un nuevo *job*, tipo responde, que se ejecuta en el nodo general para que pueda modificarse el estado del expediente para continuar con su procesamiento.

Tipo (F) – Funciones de E/S interactivas con Espera (*Por ejemplo el proceso debe esperar a que un responsable firme digitalmente uno de los documentos del expediente*).

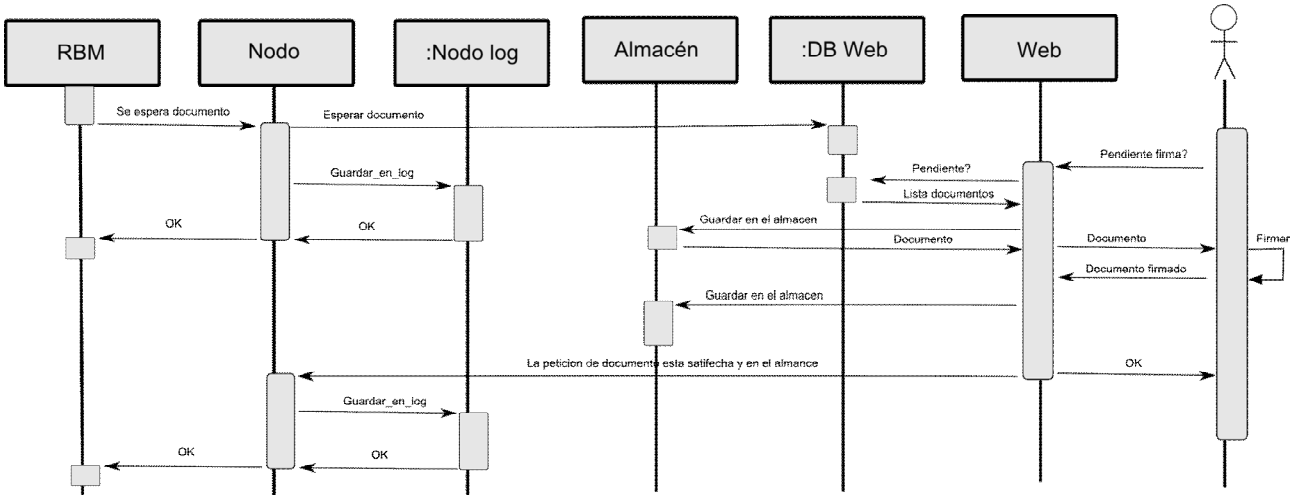


Figura 33 - Propuesta de tratamiento de tareas tipo F

Tal vez sea el procedimiento más complejo de todos pero comienza de forma similar al tipo E. Se guarda en la base de datos web el documento y la petición de la espera. Esta petición de espera será atendida por el usuario final cuando crea conveniente, introduciendo en el Almacén el documento ya firmado. Luego notificará, mediante un proceso tipo responde, que el proceso se ha finalizado. Es el trabajo tipo responde que se ejecuta en el nodo el que deja el expediente en estado para que el RBM pueda seguir ejecutándolo.

5.3.1.- Conversión de los tratamientos administrativos

Hasta el momento hemos comentado cómo debe ser el tratamiento de los trámites que componen un expediente pero estamos todavía realizando un manejo secuencial del procedimiento. Ahora estamos en disposición proceder, en la medida que el proceso lo permita, a la conversión de los flujos de trabajo de un modelo secuencial a un modelo paralelo más conveniente para su utilización en un entorno grid.

Sobre esta conversión debemos tener en cuenta numerosos textos como pueden ser la Ley de Amdahl [04] o la de Gustafson-Barsis. Con estos apoyos podemos realizar una conversión eficiente y verificar la correcta obtención de resultados. Siguiendo estos estudios se ha transformado el flujograma de trabajo expuesto en el capítulo anterior a otro más conveniente y eficiente para la arquitectura que utilizamos. Además en el gráfico se ha incorporado la valoración del tipo de tratamiento que se va ha realizar para cada uno de los trámites.

El número total de trámites coincide con el existente en la Figura 23 - Flujograma de ejemplo tratamiento de beca. El trámite “Verificar XML” corresponderá con la validación del fichero XML que tendrá la información del expediente. Este XML lo podemos ver como el contenedor de la documentación del expediente.

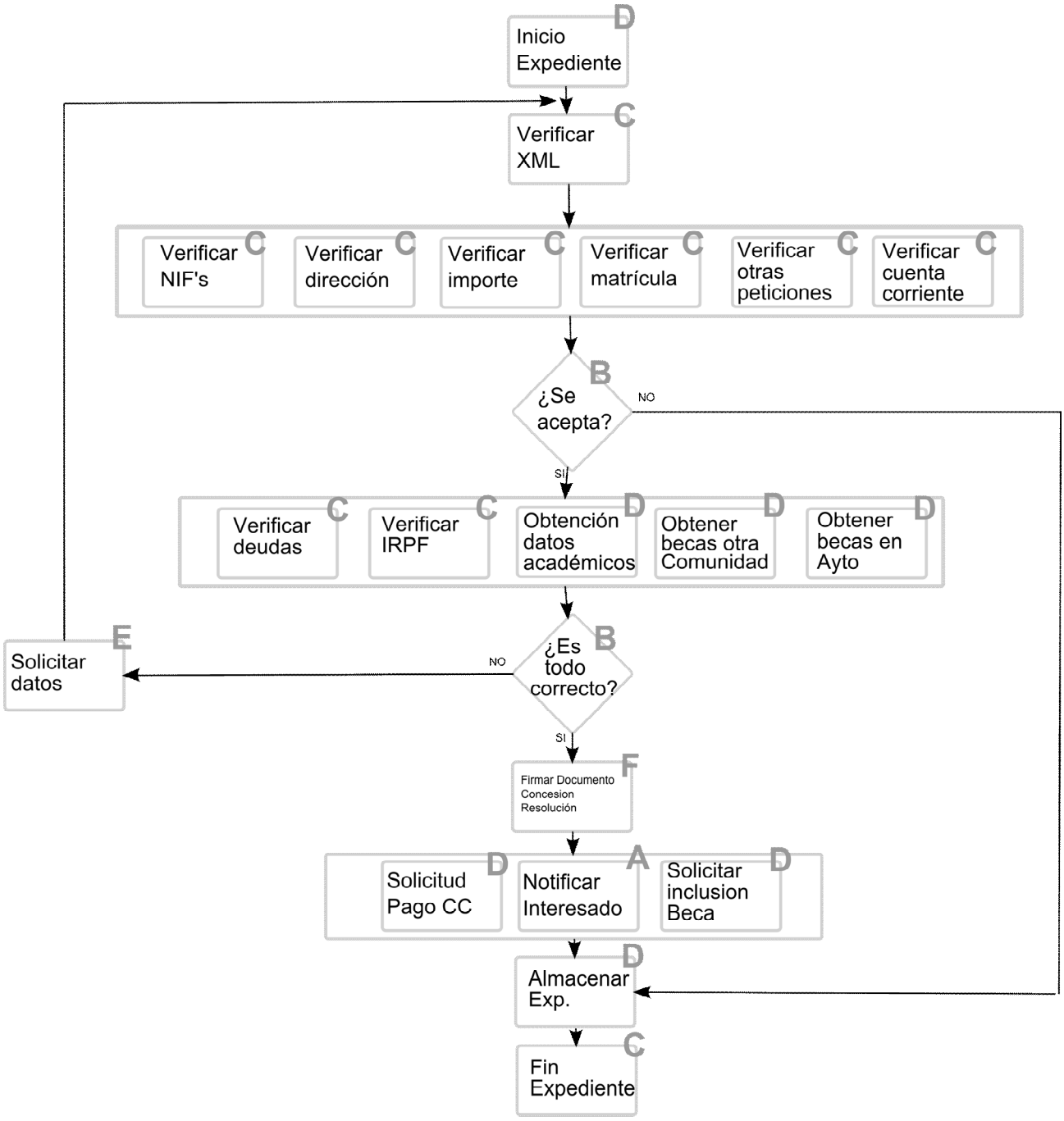


Figura 34 - Propuesta de conversión del flujograma de ejemplo

5.4.- Estrategias alternativas para el reparto de los trabajos en localizaciones remotas

En general los procesos que disponen de tecnología GRID se ven como procesos que pueden ejecutarse en cualquiera de los nodos que están disponibles. Una de las características necesarias para nuestro sistema es la posibilidad de seleccionar el/los nodos en función de la topología que se les han asignado a los trabajos. De esta forma los trabajos más especializados sólo se podrán realizar en el nodo más adecuado y que en nuestro caso, en ocasiones, es el único que lo puede ejecutar.

Para poder distribuir los trabajos se proponen dos vías de trabajo:

- Responsabilizar al balanceador de carga de la tarea de selección.
- Usar las políticas (*policies*) de JPPF.

5.4.1.- Utilización de balanceador de carga

En JPPF el balanceador de carga es la parte de la infraestructura que se encarga de la distribución de los diferentes trabajos entre los nodos disponibles. Este software es capaz de repartir los trabajos que en un momento existen disponibles para su ejecución, entre los nodos atendiendo a unos criterios previamente seleccionados.

Para realizar esta tarea se dispone de dos tipos de información: Información proporcionada por los nodos y los metadatos procedentes de los trabajos.

Para introducir información adicional en los nodos hay que crear un nuevo Mbean que proporcione información del nodo o bien modificar el fichero de configuración del nodo para que incluya nueva información.

La introducción de metadatos en los trabajos es una labor sencilla que se realiza modificando el fichero de propiedades de las tareas.

Para modificar el balanceador de carga y que acepte estas nuevas variables hay que crear un balanceador de carga modificado. El Framework JPPF permite la creación de balanceadores de carga en función del tipo de tarea que se debe realizar. Esto no deja de ser complejo y existen muchos algoritmos diferentes. El documento [11] describe las diferentes vías que existen para realizar tareas de balanceo de carga en los sistemas grid.

5.4.2.- Asignación de grupos en los nodos

Otra posible vía es la utilización de las políticas de ejecución que JPPF tiene activadas en el balanceador de carga por defecto. Para realizar la asignación en función de las SLA (*Service Level Agreements*) es necesario realizar las siguientes tareas:

1.- Modificación del fichero de configuración (jppf-node.properties) de los nodos. Se añade algo parecido a la línea del cuadro siguiente en función del desarrollo jerárquico descrito en el punto 5.2.1

```
group.valida.dni=1
```

y que indica que el nodo pertenecerá al grupo *valida.dni*. Se podría desactivar eliminando la línea, comentándola (#) o indicándole un valor 0.

2.- Introducir la política de ejecución únicamente en los nodos con un metadata determinado. Para ello es necesario incluir en la ejecución del código sentencias del tipo siguiente (ver también el anexo 3):

```
ExecutionPolicy LessThan2ThreadsPolicy = new MoreThan("processing.threads", 0);
job.getJobSLA().setExecutionPolicy(LessThan2ThreadsPolicy);
```

Esta segunda solución es la que se utilizará para la implementación que se realizará en los siguientes capítulos.

5.5.- Ciclo de vida de los trabajos

Si seleccionamos un determinado instante no pueden estar todos los expedientes en un estado de ejecución. Dependiendo de los factores externos los expedientes podrán estar en diferentes estados que le permitirán o no su ejecución en los nodos del sistema. Es el *Record Broker Manager* (RBM) el proceso que se encargará de ponerlos en funcionamiento y gestionar los diferentes estados de los procesos en función de las notificaciones que le lleguen.

Tengamos en cuenta que los estados de ejecución son diferentes de los estados de tramitación. Los estados de tramitación se definen funcionalmente con el diseño del flujo de información y dependerán de los datos que podamos obtener en cada momento.

Se proponen los siguientes estados que se tratarán para lograr un funcionamiento básico.

- Listo. Se define este estado cuando se recibe algún tipo de acción de creación de expediente o bien cuando el sistema lo deja dispuesto para que se pueda ejecutar.
- En ejecución. Se encuentra en algún nodo funcionando o pendiente de la asignación del nodo.
- Bloqueado. Espera algún tipo de actuación o introducción de datos externos.
- Finalizado. Se ha terminado la ejecución del flujo de información.

Un expediente llegará al sistema y se encontrará en el estado "**Listo**". Pasará a ejecutarse cuando el planificador de los expedientes (RBM) genere los trabajos para la ejecución en los nodos, en ese momento se encontrará en estado "**En Ejecución**". Cuando solicite algún trabajo de entrada o salida del sistema de archivo (tipos de tareas E o F) pasará al estado de "**Bloqueado**". De este estado saldrá cuando un servicio lo extraiga, de forma similar a un interrupción del SO. Pasará a estado "**Finalizado**" cuando terminen todos los *job* y el *job* principal.

En [49] podemos ver un ejemplo de tratamiento de expedientes por medio de los estados de los procesos. El modelo utilizado se denomina ciclo de vida de los expedientes y por lo tanto utilizaremos este término para implementar nuestra propuesta.

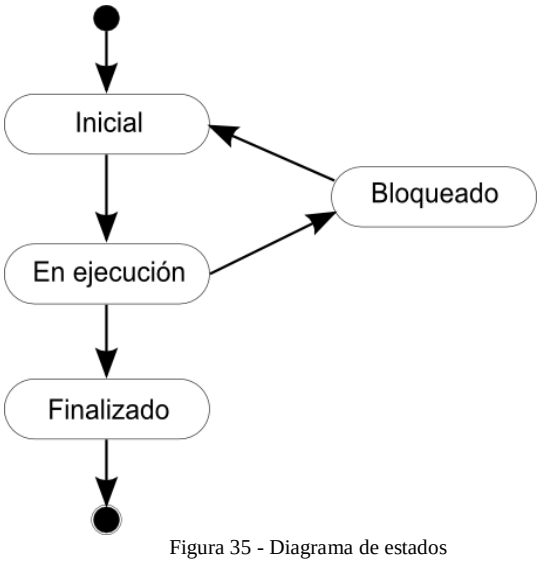


Figura 35 - Diagrama de estados

Parte IV - Prototipo de implementación

6 .- Definición del prototipo

Durante los capítulos y secciones anteriores se han plasmado las estrategias para la posible conversión de los sistemas de workflow secuenciales a un nuevo modelo basado en la ejecución en paralelo de las tareas basado en una tecnología grid. En éste y en el próximo capítulo se intentará realizar una primera aproximación a la implementación con el fin de verificar las peculiaridades del entorno propuesto.

En el presente capítulo se definirá la estructura del prototipo, la distribución de los servicios de ejemplo que se está utilizando durante el trabajo y se realizará un análisis/diseño del código necesario para el prototipo.

Con esto se pretende conocer un poco más las estrategias de trabajo que deben utilizarse para una posible implementación en un entorno real. El ejemplo que se utiliza se introduce en las diferentes problemáticas del uso de los servicios de cómputo remotos que un sistema grid permite manejar.

6.1.- Arquitectura lógica

Hemos visto que podemos utilizar una infraestructura grid para el manejo del workflow administrativo, pero queda pendiente definir una estructura lógica que mantenga y de soporte al funcionamiento deseado. La propuesta de arquitectura lógica se puede concretar en el siguiente esquema donde se definen los diferentes componentes de la infraestructura.

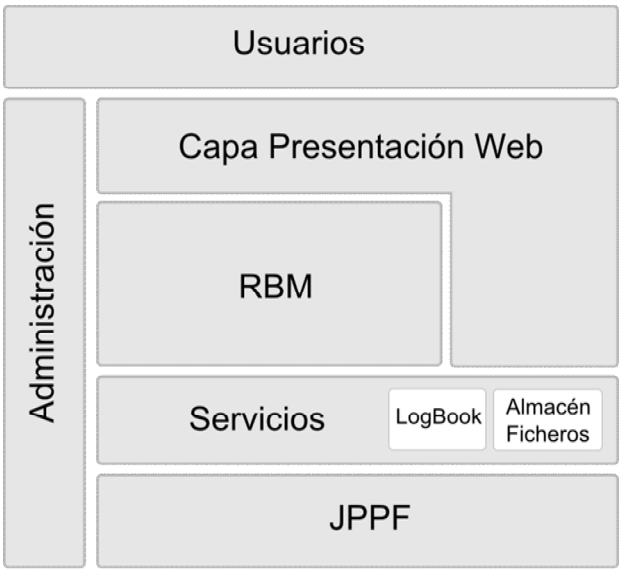


Figura 36 - Propuesta de arquitectura para la implementación de la infraestructura

Podemos ver los diferentes módulos que completan el sistema:

- **Usuarios**. Usuarios finales que utilizarán la aplicación.
- **Capa de presentación Web**. Sirve de interfaz entre la infraestructura y los usuarios. Debe contener su propia base de datos para almacenar la información requerida. Debemos recordar que JPPF tiene un conector J2EE que le confiere la funcionalidad de poder acceder a la realización de tareas en el GRID.
- **Gestor de Expedientes** (RBM – *Request Broker Manager*). Se encargará de introducir los expedientes para su ejecución dentro de la infraestructura. La fuente de datos de este gestor es un almacenamiento permanente. De esta forma, se asegura el número total de expedientes en ejecución en un instante determinado o el estado en el que se encuentran. El gestor de expedientes no es el encargado de la ejecución de los programas asociados a los expedientes sino de introducir un mayor o menor número de expedientes en el framework.
- **Servicios**. En esta capa podemos encontrar todos los servicios que se pueden incluir dentro de la infraestructura. Entre los más destacados podemos encontrar el almacén de ficheros que permite guardar de forma permanente los ficheros de datos y el LogBook que proporciona almacenamiento de las incidencias que se deseen guardar.
- **Administración**. Están incluidas aquí todas las herramientas de administración para la capa Web, el RBM, los servicios y especialmente la consola de administración de JPPF que controla el funcionamiento de los Drivers y Nodos existentes.

6.2.- Definición de nodos

Para poder realizar las pruebas se definirán los siguientes nodos: Tres serán máquinas físicas y dos serán máquinas virtuales con las mismas funcionalidades que las otras tres. Se usará tanto el sistema operativo *MS Windows* (2003 o XP) como Linux (*RedHat* o *Ubuntu*).

Nombre del nodo	Abreviatura	Máquina
Gobierno autonómico 1	ga1	Envy
Gobierno autonómico 2	ga2	Greed
Gobierno nacional	gn	Lust
Ayuntamiento 1	ayto1	Mv1
Ayuntamiento 2	ayto2	Mv2

Tabla 7 - Definición y emplazamiento de nodos

Todas las pruebas y desarrollos que se van a realizar, en este capítulo y en el posterior, estarán referenciadas a este hardware. El prototipo pretende simular una estructura real de trabajo e interconexión de entidades. En este caso se ha diseñado sobre un total de cinco administraciones conectadas de forma remota para conseguir el tratamiento de un expediente. Podemos situar los nodos en un mapa para resaltar la situación que se desea reproducir y aclarar el entorno de simulación que se pretende lograr. En este esquema no se pretende indicar la centralización, sino la distribución de los trabajos por parte de un generador de los mismos y la localización remota de los nodos que servirán las peticiones.

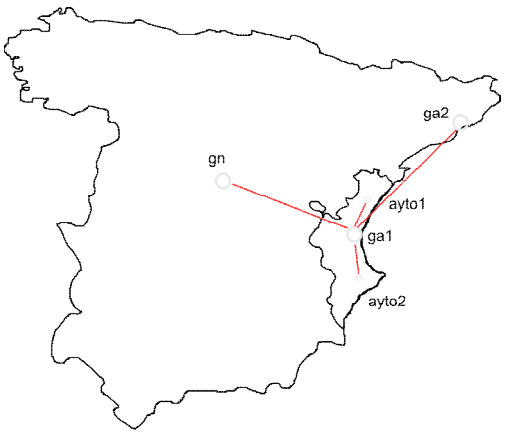


Figura 37 - Diagrama explicativo del diseño

6.3.- Reparto de servicios

Se pondrán en marcha los siguientes servicios que son los necesarios para la ejecución del workflow descrito en el capítulo cinco. Estos se extraen del diagrama de flujo de trámites a realizar que se muestra en la Figura 34 - Propuesta de conversión del flujograma de ejemplo existente en el punto 5.3.1.

Trámite	Servicio	Tipo
Inicio expediente	ga1.inicio_expediente	D
Verificar XML	ga1.verifica_xml	C
Verifica NIF	gn.verifica_nif	C
Verifica Dirección	gn.verifica_dir	C
Verifica Importe	ga1.verifica_importe_beca	C
Verifica Matricula	ga1.verifica_matricula	C
Verifica Otras Peticiones	ga2.verifica_peticiones	C
	ayto1.verifica_peticiones	C
	ayto2.verifica_peticiones	C
Verifica CC	ga1.verifica_cc	C
¿Se Acepta?	ga1.se_acepta.solicita	B.fase_sol
	ga1.se_acepta.responde	B.fase_resp
Verifica Deuda	gn.verifica_deuda	C
Verifica IRPF	gn.verifica_irpf	C
Obtener Datos Académicos	gn.obtener_datos_academicos	D
	ga1.obtener_datos_academicos	D
	ga2.obtener_datos_academicos	D
Obtener beca otra comunidad	ga2.becas_otra_comunidad	D
Obtener beca Ayto	ayto1.becas_ayto	D
	ayto2.becas_ayto	D
¿Es correcta?	ga1.es_correcto.solicita	B.fase_sol
	ga1.es_correcto.responde	B.fase_resp
Firmar Documento	ga1.firma_documento.solicita	F.fase_sol
	ga1.firma_documento.responde	F.fase_resp
Solicitar Pago	ga1.solicita_pago	D
Notificar Interesado	ga1.notificar_interesado	A
Solicitar Inclusión PNB	gn.solicitar_pnb	D
Almacenar Expediente	ga1.almacenar_expediente	D
Solicitar Datos	ga1.solicitar_datos.solicita	E.parte_sol
	ga1.solicitar_datos.responde	E.parte_resp
Fin Expediente	ga1.fin_expediente	D

Tabla 8 - Relación de servicios por trámite

En esta tabla no está especificado el servicio necesario para la grabación de los log. Consideraremos que todos los procesos realizan, tanto en el momento del inicio de su ejecución como en la finalización del proceso, la grabación de una línea en el log llamando al servicio general.logbook. Este servicio es parte de la infraestructura.

El reparto de los servicios por topología y servidor serán los indicados en la tabla siguiente. Los anexos a los ficheros de configuración que se han utilizado se encuentran detallados en el Anexo 6.

Entidad	Gobierno Autonómico 1	Gobierno Autonómico 2	Gobierno Nacional	Ayto1	Ayto2	General
Tipo A	ga1.notificar_interesado					
Tipo B	ga1.es_correcto.solicita ga1.es_correcto.responde ga1.se_acepta.solicita ga1.se_acepta.responde					general.logbook
Tipo C	ga1.verifica_importe_beca ga1.verifica_matricula ga1.verifica_cc ga1.verifica_xml	ga2.verifica_ peticiones	gn.verifica_nif gn.verifica_dir gn.verifica_deuda gn.verifica_irpf	ayto1.verifica_peticiones	ayto2.verifica_peticiones	
Tipo D	ga1.inicio_expediente ga1.obtener_datos_academicos ga1.solicita_pago ga1.almacenar_expediente ga1.fin_expediente	ga2.obtener_datos_academicos ga2.becas_otra_comunidad	gn.obtener_datos_academicos gn.solicitar_pnb	ayto1.becas_ayto	ayto2.becas_ayto	
Tipo E	ga1.solicitar_datos.solicita ga1.solicitar_datos.responde					
Tipo F	ga1.firma_documento.solicita ga1.firma_documento.responde					

Tabla 9 - Topología y localización de los servicios implementados

6.4.- Análisis del software para las pruebas

En este punto se tratarán cuestiones de análisis del sistema que son relevantes ya que caracterizan bien a la implementación en JPPF o bien por su relación con el tratamiento de la información en un entorno de computación paralela.

Algunos de los trámites diseñados necesitan alguna actuación externa. Si deseamos realizar pruebas que permitan la extracción de datos estadísticos, el modelado de estos trámites deberá seguir un patrón diferente e implementarse con respuestas automáticas. Los trámites que suponen la petición de datos, la introducción de ficheros o la interacción con usuarios finales se automatizarán de tal forma que no sea necesaria una actuación externa con el fin de que su existencia no altere los resultados y los tiempos de respuesta del resto de los trámites y componentes del sistema diseñado.

6.4.1.- ¿Qué se desea implementar?

Nuestro objetivo ha sido verificar que el workflow administrativo es capaz de diseñarse e implementarse en un entorno grid. Por ello, no se va a realizar una implementación de todos los servicios, sólo de los "contenedores" de estos servicios. Se evaluará que las suposiciones realizadas durante los capítulos anteriores pueden ponerse en marcha e incluso pueden llegar a ser utilizadas en un sistema productivo. De esta forma, la suposición de que un trámite verifica un dato determinado poseerá una respuesta similar a la del sistema productivo, real aunque la consulta que se realice no suponga acceso a datos locales en cada uno de los nodos.

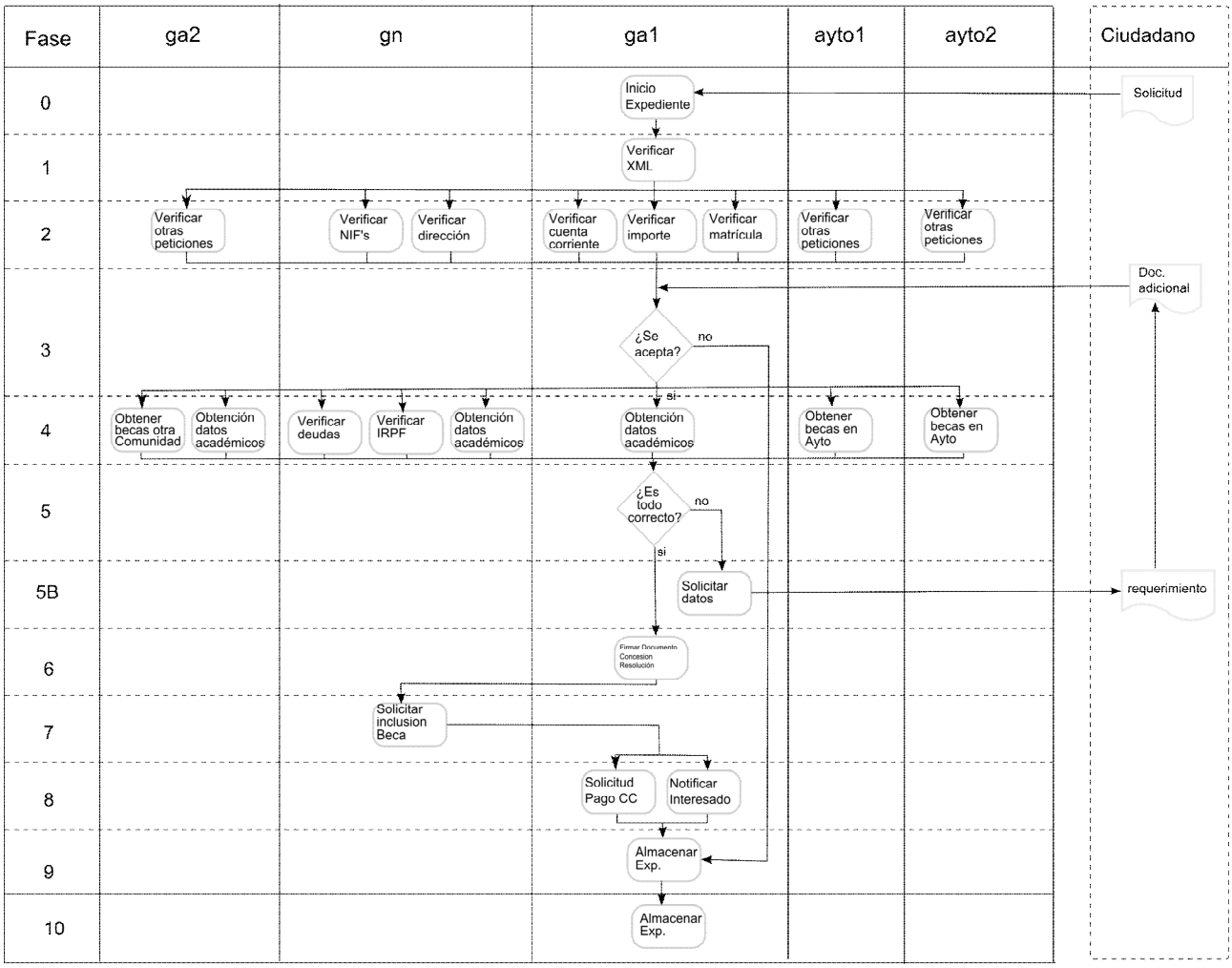


Figura 38 - Diagrama de ejecución

El diagrama anterior puede aclarar qué servicios deben ejecutarse en paralelo y en por qué nodo realizarán la tarea. Este esquema tiene su equivalente en la Figura 24 - Representación gráfica del procedimiento administrativo donde se muestran los trámites y el lugar donde se realiza. En nuestro caso el lugar de tramitación se ha sustituido por el lugar de ejecución.

6.4.2.- Clases

El diagrama de clases simplificado del código desarrollado quedará de la siguiente forma:

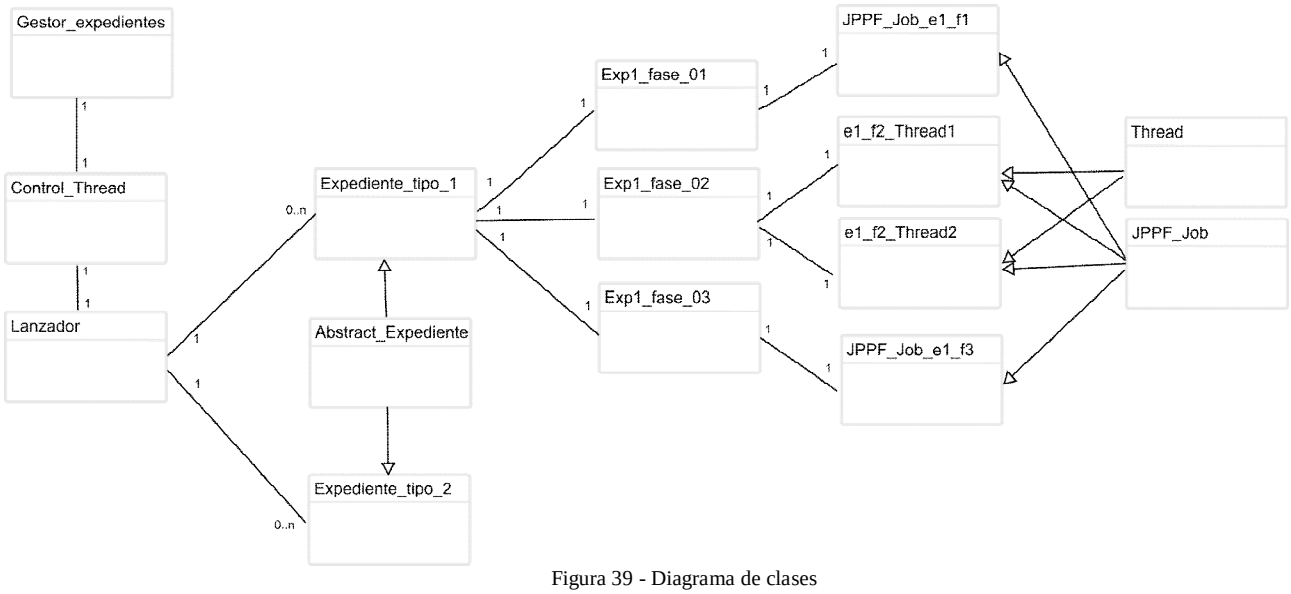


Figura 39 - Diagrama de clases

El diagrama se ha simplificado con el fin de mostrar aquellas clases que son realmente interesantes dentro del desarrollo. La clase Control_Thread es la que hace la función de RBM e intenta mantener la carga de la infraestructura al nivel indicado en el fichero de configuración. Expediente_tipo_1 o tipo_2 son dos implementaciones de posibles expedientes administrativos. Estos poseen sus fases Exp1_fase_01, Exp1_fase_02 y en cada una de las fases se pueden ver los Job JPPF que se ejecutan en los diferentes nodos. La ejecución del expediente 1 es la que se utilizará para todas las pruebas y la que se ha implementado con la estructura de la tramitación de ejemplo de beca.

6.4.3.- Otros temas a destacar

Seguridad. En varias ocasiones se ha expuesto que el entorno de desarrollo seleccionado carece de las características de seguridad necesarias para considerarlo como un entorno seguro. Sólo se realizará la mención de las posibles transferencias seguras que se permiten en los nodos como parte del framework, ya que no se ha diseñado ninguna característica adicional que se pueda destacar como suplemento a lo poco ya existente.

El tiempo dentro de los nodos. Existen varios protocolos para poder ajustar el tiempo en diferentes nodos al indicado por un servidor central (NTP – Network Time Protocol [50] o SNTP – Simple NTP [51]) . Para nuestro caso, el tiempo que se utilizará para las posibles mediciones será el que indique el nodo general.logbook. De esta forma se considera que la remisión a este nodo del trabajo es parte de la ejecución de la tarea y en todos los casos se utilizará la misma desviación.

6.5.- Diseño del software para las pruebas

Para poder realizar el ensayo se ha desarrollado nuevo software que cumple el propósito de realizar las pruebas necesarias de funcionalidad del entorno. Se ha estructurado en tres partes. La primera parte es un generador de expedientes aleatorios para poder realizar pruebas de carga y estrés del sistema. La segunda, en RBM que se encarga de obtener los expedientes e introducirlos en la infraestructura para que se puedan ejecutar. La tercera parte es la porción de código que simula los diferentes servicios que se ejecutan en cada uno de los nodos. La funcionalidad de los servicios de los nodos no se ha implementado ya que no es significativa para las pruebas que se ejecutarán. El código está codificado en java y se ha encapsulado en dos jars diferenciados. Uno para la creación de trabajos y otro para el resto de la aplicación. Todo el código está desarrollado con el entorno de desarrollo Netbeans [53]. Se ha utilizado HyperSQL como base de datos para el almacenamiento de la información necesaria del RBM. Se distribuye bajo una licencia libre y está enteramente implementada en Java.

De todos los trabajos que se pueden realizar en una fase de diseño del software, se destacarán aquellas que parecen más importantes a la vista del trabajo realizado de implementación.

Los casos que se tratan a continuación han sido implementados a modo de ejemplo para evaluar su viabilidad dentro del entorno grid. Una implementación total podría tener una extensión no deseada del proyecto.

6.5.1.- Diseño de la base de datos

Como resultado de la conversión del diagrama de entidad-relación a modelo relacional se ha realizado la codificación utilizando SQL. El resultado se puede consultar en el Anexo 5.

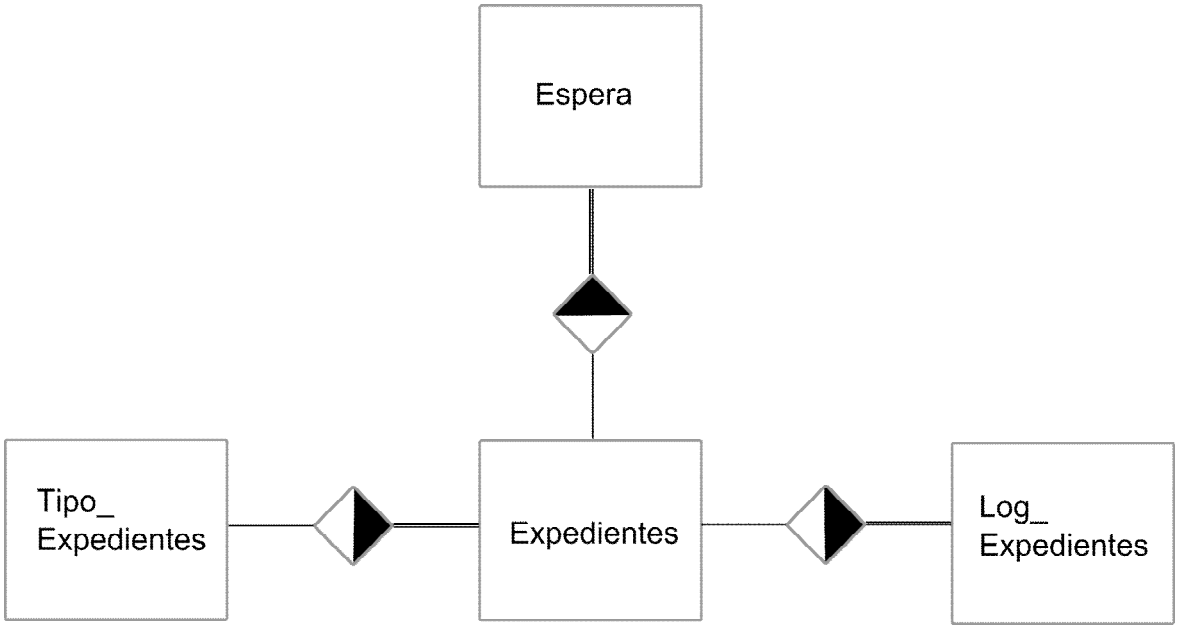


Figura 40 - Modelo E-R

En la entidad Expedientes se almacenará información propia de los mismos (código único, estado, etc.). La entidad Log_Expedientes podrá almacenar información sobre el log de la ejecución de los expedientes. Tipo_Espedientes indicará las características del expediente genérico que se almacena. En la entidad Espera se guarda la información de los eventos de espera que deben tener almacenados hasta su aparición o expiración.

6.5.2.- Implementación de los estados de ejecución

Al final del capítulo 5 se definían los estados en los que se podía encontrar un expediente independientemente de la fase de ejecución que estuviera tramitando. Para poder llevar estos estados de ejecución a la implementación se ha definido los siguientes criterios para conocer si un expediente está en un estado o en otro.

- Estado Inicial. Se encuentra con STATUS=1 en la tabla de expedientes.
- En ejecución. Se encuentra con STATUS=2 en la tabla de expedientes.
- Bloqueado. Se encuentra con STATUS=2 en la tabla de expedientes y además tiene al menos una línea en la tabla espera con el código de la petición asociada.
- Finalizado. Se encuentra con STATUS= 99.

Pero también es necesario que se almacene información sobre las fases de la tramitación y especialmente en la que se encuentre en un momento dado el expediente. Se ha definido un campo FASE en la tabla Expedientes que es actualizado mientras se avanza en la tramitación.

6.5.3.- Paso de datos a los servicios

Una de las cuestiones que debemos tratar un poco más en profundidad es la vía en que la información llega a los distintos programas que están en ejecución dentro del framework.

La distribución en localizaciones remotas soluciona el problema del acceso a los recursos que se consideran remotos pero que son locales a la ejecución por los nodos de las Tareas y/o Jobs que se les remiten. Este acceso a bases de datos o ficheros estará regido por los permisos que posea cada uno de los entornos y se podrán manejar en función de las capacidades de acceso que se habiliten al entorno de ejecución Java (JVM) más los procedimientos locales de seguridad.

Por otro lado tenemos la utilización de información común para varios procesos y que es común a toda la infraestructura o simplemente a la ejecución de un expediente. Esta información, que deberá fluir hasta los nodos se plantea en situaciones diferenciadas.

- **Transferencia de ficheros:** Hasta el momento no se ha tratado el método para el paso de los datos a los servicios. Si se ha comentado que la forma que utilizará el servicio de almacenamiento se basará en ftp, pero no se ha comentado cómo los servicios y programas van a transferir la información entre los diferentes objetos que se encuentran en un momento dado en ejecución.
  Existen numerosos métodos para poder hacer llegar los ficheros hasta los programas en ejecución. Los frameworks descartados en el capítulo 2 poseen, en general, un sistema de para compartir ficheros. GridFTP es una implementación para Globus Toolkit, HDFS de Apache Hadoop pero también se puede utilizar SFTP. No se entrará a analizar todos estos protocolos ya que cualquiera de ellos nos sería de utilidad siempre que exista o seamos capaces de realizar código java para su utilización.

- **Transferencia de variables del programa**: JPPF proporciona mecanismos de transferencia de información de forma interna a la infraestructura.

- **Servicio de grabación y selección único de ficheros.** Con esta idea nos mantenemos fieles a la propuesta de trabajo en grid, derivando el trabajo propio de guardar la información a un servicio. Este servicio puede ser implementado por diferentes tipos de soluciones, bien por medio de un servidor de ficheros, como los indicados anteriormente, un sistema de ficheros local o un sistema de tratamiento documental estilo Alfresco [02] que permitiría un mejor acceso y tratamiento de los ficheros.

6.5.4.- Concurrencia en la utilización de datos

La utilización de servicios en paralelo junto con información común crea algunos problemas que es necesario tratar antes de continuar con la implementación. Hasta el momento no se ha indicado nada sobre la capacidad de compartir datos entre los diferentes procesos existentes de forma concurrente.

Los problemas que surgen son similares a la problemática de la ejecución concurrente de varios procesos y la compartición de datos. Semáforos, monitores o regiones críticas son algunas de las soluciones para estos problemas, pero podemos incluir otra que sería la utilización de tickets proporcionados por un servicio común para evitar el tratamiento concurrente de un determinado dato/fichero/recurso. Este proceso, que evitaría el problema de la concurrencia en la utilización de datos, debe ser capaz de autorizar un proceso mientras deniega a los demás procesos el acceso. Puede tratarse como un proceso de tipo E o F lo que permitiría la espera de los procesos mientras se liberan los datos por el nodo que lo está tratando.

6.5.5.- Ejecución de tareas en los nodos

Las tareas que se ejecutan en cada uno de los nodos son programas realizados para ejecutar exclusivamente una tarea local dentro de un entrono productivo local. JPPF expone como posibilidades las siguientes:

- POJO. (*Plain Old Data Object*) que permite la ejecución de programas especialmente diseñados en entorno, generalmente con origen en otros lenguajes de programación. También es óptimo cuando no se dispone del código fuente original de la aplicación o no se puede alterar.
- Lanzar tareas desde la línea de comandos. Esta opción permite la ejecución de programas dentro de los JPPFTask como si estuviéramos lanzándolas desde la línea de comando.
- Cargador de Clases locales que se encuentren el PATH de ejecución del nodo.
- Programar la clase dentro del programa general.

Las tres primeras opciones permitirían no tener que transmitir el programa por medio de la red en el momento de la asignación.

Si se desea la ejecución de un proceso remoto incluido dentro del propio paquete, éste deberá ser enviado a cada uno de los nodos donde se quiera ejecutar. Esto puede ser un buena opción si los procesos cambian generalmente o bien son pequeños. También puede ser incorrecto que se encapsule todo dentro de un paquete con un tamaño elevado que se tramitará con excesivo coste.

El coste de la ejecución de las dos primeras opciones es superior en el caso de tareas repetitivas y sin modificaciones.

7 .- Puesta en marcha del prototipo y pruebas

En este capítulo se describe como se ha puesto en marcha y las pruebas que se han ejecutado, para verificar de correcto funcionamiento del prototipo. Debemos tener en cuenta que es un prototipo con limitaciones, principalmente en sus capacidades de ejecución ya que el número de núcleos disponibles ha sido muy reducido.

El fin principal será analizar los servicios que han sido distribuidos entre las diferentes máquinas físicas y virtuales. Así se asegura su correcto funcionamiento dentro de una estructura colaborativa donde todos los servicios se ponen a disposición de un único fin, la gestión común de los expedientes.

7.1.- Puesta en marcha del prototipo

En la imagen siguiente podemos ver la consola de monitorización de la infraestructura con los diferentes nodos conectados. Se pueden visualizar las 5 máquinas (entre virtuales y físicas) que componen todo el prototipo de pruebas.

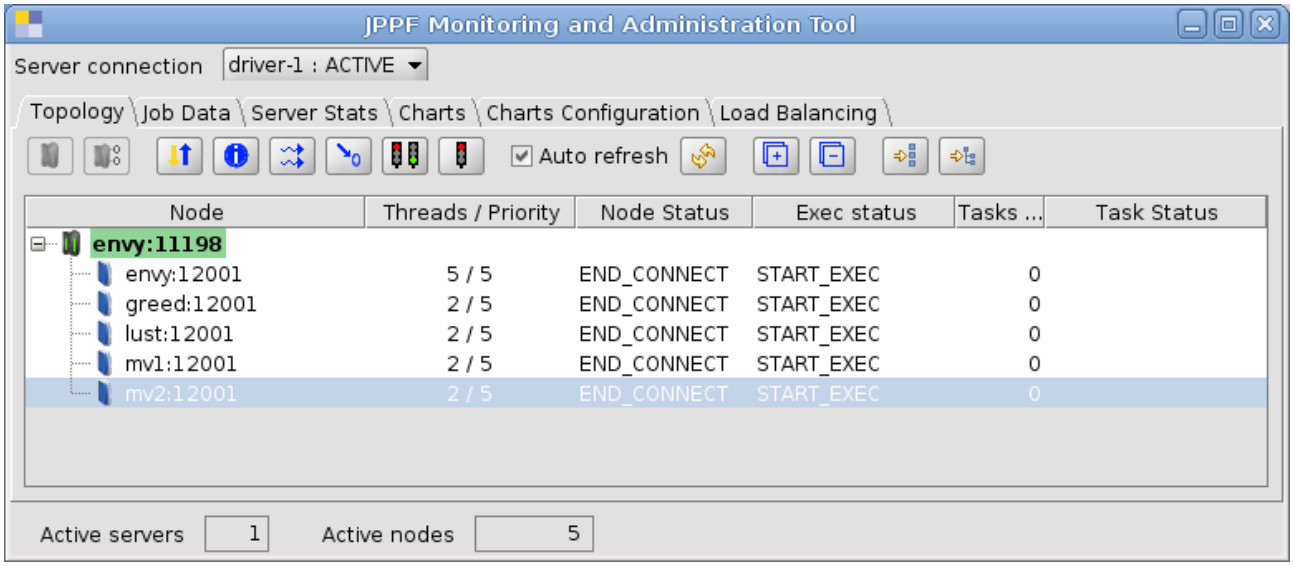


Figura 41 - Captura de la infraestructura para las pruebas

La configuración de cada uno de los nodos se puede ver en el Anexo 6 donde se muestran los servicios que se van a prestar en los diferentes nodos. Estos servicios, que habían sido definidos anteriormente, se han trasladado al fichero de configuración.

El proceso RBM (*Request Broker Manager*) correrá en una de las máquinas, no siendo importante en cual. De hecho para facilitar la tarea de búsqueda dentro de la red se ejecutará en el mismo nodo que el Driver.

Como opciones de configuración del Driver se han utilizado las proporcionadas de forma estándar con los módulos en la página web de JPPF. De la misma forma, se han mantenido en todos los nodos la configuración por defecto proporcionada con el fin de no alterar los resultados de las pruebas del prototipo.

Los servicios se han implementado de forma ficticia, esto es, el código que ejecuta no es importante ya que depende de la implementación que tenga en cada uno de los lugares donde se plantea el servicio. Para simular el tiempo de espera se ha realizado una función de espera aleatoria de tiempo que permite simular el tiempo de proceso. La respuesta siempre será la correcta, no existe la posibilidad de que se produzca un error en la ejecución del servicio.

La utilización de este tiempo de espera aleatorio se ha configurado como un parámetro global y permite su modificación en cada ejecución. Para no alterar los resultados de los datos extraídos se ha utilizado un valor de espera 0 lo que hace que no influya en su ejecución ni que el prototipo tenga una carga de trabajo excesiva.

El LogBook realiza la grabación en un fichero de log que se sitúa en el nodo designado para el servicio. Este nos servirá para poder realizar labores de monitorización de las tareas realizadas y de tiempo para los eventos del sistema. Se ha descartado al final la utilización de la tabla log_expedientes para evitar sobrecargas de los nodos no necesarias.

La función de obtención y guardado de ficheros de la infraestructura se ha implementado mediante un cliente ftp. Simplemente se ha verificado que funciona correctamente y que en cualquier momento se puede acceder a los datos de un expediente.

En los procesos B y F que tienen como principal motivo de existencia la interacción con el usuario, en los casos de evaluación de los tiempos del sistema serán sustituidos por una respuesta automática. Esto permite no tener una fuente de espera aleatoria que podría alterar la toma de tiempos.

7.2.- Realización de las pruebas

Para la realización de las pruebas se ha generado una carga artificial de trabajos que se introducirán en el sistema por medio de un programa java. La carga de datos es continua y no existen tiempos de espera entre dos inserciones que no sean los propios de la ejecución del programa.

7.2.1.- Diseño de las pruebas y resultados previstos

Para la realización de las pruebas del prototipo vamos a utilizar un esquema similar a las pruebas que se han detallado en la sección 3.3 y posteriores.

- **Grupo A.- Pruebas de funcionamiento básico.**

  - A1.- Puesta en marcha de trabajos básicos con ejecución paralela en varios nodos. En estas pruebas se pretende asegurar que cada uno de los trabajos se realiza en la localización esperada. De esta forma será lógico que en un trabajo tipo se ejecuten los siguientes trabajos por nodo.

Nodo	Correspondencia	# Jobs previstos en el nodo	#Jobs previstos para logbook	#Jobs total
Envy	Gobierno autonómico 1	13	0	13
Greed	Gobierno autonómico 2	3	22	25
Lust	Gobierno nacional	6	0	6
Mv1	Ayto 1	2	0	2
Mv2	Ayto 2	2	0	2

Tabla 10 - Ejecuciones previstas

Después de la ejecución de un trabajo, el total de trabajos ejecutados en el nodo deberá ser el indicado. Para ello se inicializarán los contadores antes de lanzar la ejecución de un único expediente.

- A2.- Pruebas de puesta en funcionamiento de varios servidores (Drivers). No se va a realizar esta prueba ya que hemos visto que no tiene ningún resultado en el tiempo de ejecución de los trabajos.

- **Grupo B.- Evaluación de las características propias del GRID.**
  - B1.- En trabajos con gran utilización del sistema, verificar el balanceo de carga. En este caso vamos a duplicar los nodos para poder ejecutar varios expedientes de forma simultánea. La ejecución con el doble de nodos debe suponer una mejora de los tiempos de ejecución, pero en nuestro caso solo se evaluará el reparto de la carga entre los nodos que poseen la misma responsabilidad.
  - B2.- En trabajos extensos, verificar el comportamiento del sistema ante caída de nodos. Sobre la realización anterior de dos nodos por servicio se deben suprimir algunos servicios para verificar que el servicio restante realiza las tareas que deberían hacer los dos en conjunto.
    En este caso pararemos uno de los nodos duplicados del caso anterior y verificaremos que todo termina correctamente.
  - B3.- En trabajos extensos, verificar el comportamiento del sistema ante la recuperación de nodos. Comprobar que cuando se levantan otra vez los nodos, vuelven a ejecutar los trabajos deseados. En nuestro caso después de parar el nodo lo volveremos a poner en marcha para verificar que cumple con su cometido.
  - B4 y B5 - En trabajos extensos, verificar el comportamiento del sistema ante caídas y recuperaciones de servidores (Drivers). Se prevé que el funcionamiento sea el mismo que el descrito en las pruebas realizadas anteriormente en el capítulo 3. Por lo tanto no se realizarán aquí.
- **Grupo C.- Pruebas de estrés del sistema (carga).**
  - C1.- Ejecución de trabajos muy intensivos en cómputo divididos en numerosos trabajos. Los procesos diseñados no tienen un coste computacional elevado, por lo tanto dejamos las pruebas de carga para la prueba C2.
  - C2.- Ejecución de trabajos muy extensivos en tiempo divididos en numerosos trabajos. Se pretende evaluar la capacidad del sistema de ejecutar numerosos procesos.
  - C3.- Ejecución de trabajos mezclados tanto en tiempo como con número de trabajos. Por la misma razón que el C1 no se realizarán.

- **Grupo D.- Pruebas de escalabilidad del sistema.**
  - D1.- Verificar la mejora de tiempos con el funcionamiento de gran cantidad de nodos. Se utilizarán dos nodos para cada uno de los servicios, debiendo aparecer una mejora de los tiempos de ejecución debido a una ejecución paralela de los trabajos.
  - D2.- Verificar si existe mejora con la introducción de varios Drivers. Los resultados esperados son los mismos que los obtenidos en el capítulo 3 por lo tanto no procede la utilización de varios Drivers.

- **Grupo E.- Comportamiento del interface gráfico de usuario.** Una vez verificado el funcionamiento en el capítulo 3 no tiene sentido que sigamos evaluando el comportamiento del interface gráfico. Por lo tanto ni E1 ni E2 se ejecutarán.

7.2.2.- Resultados de las pruebas

Los resultados para cada una de las pruebas ha sido el siguiente:

Grupo A.- Pruebas de funcionamiento básico.

A1.- Prueba de funcionamiento básico con la ejecución en paralelo de los trámites. El total de los trabajos observados y de los previstos coincide como se había supuesto. Como se puede observar en la figura y la tabla siguientes se ve que los trabajos se han distribuido correctamente entre los diferentes nodos que están disponibles.

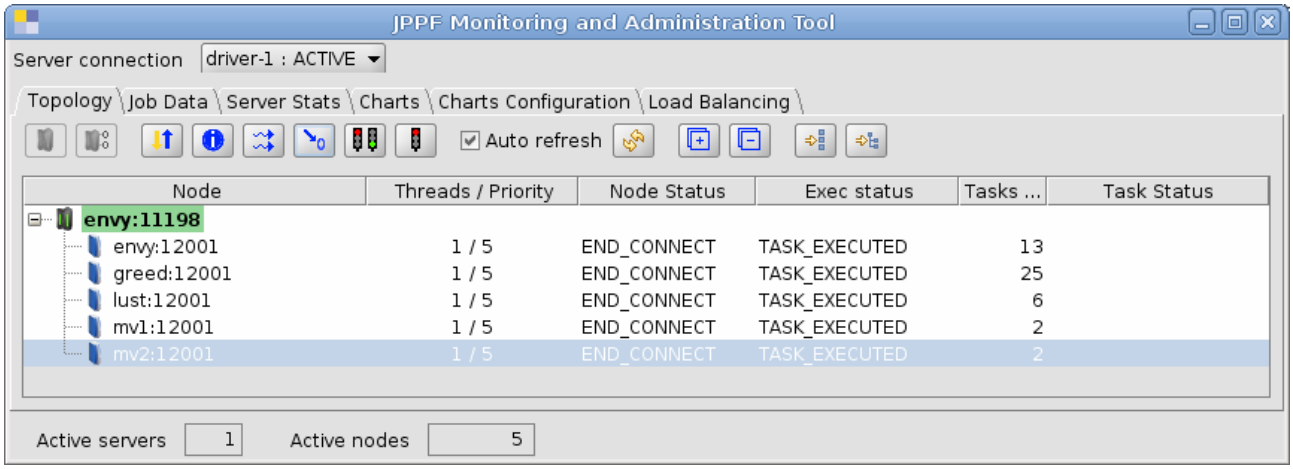


Figura 42 - Trabajos realizados en cada nodo para un expediente

Nodo	Correspondencia	# Jobs previstos en el nodo y observados
Envy	Gobierno autonómico 1	13
Greed	Gobierno autonómico 2	25
Lust	Gobierno nacional	6
Mv1	Ayto 1	2
Mv2	Ayto 2	2

Tabla 11 - Ejecuciones observadas

A2 .- No procede

Grupo B.- Evaluación de las características propias del GRID.

B1.- Verificación del Balanceo de Carga. Tras ejecutar las tareas se comprueba que existe un balanceo de la carga. Esta verificación se ha realizado poniendo los contadores a cero y ejecutando un único expediente.

Este caso es diferente al mostrado en el apartado 3.5. En la prueba actual todos los nodos tienen la misma capacidad de finalización de los trabajos (la productividad de cada uno será igual). Los procesos ejecutados no tienen necesidad de procesamiento por lo tanto no depende de las capacidades del procesador. Con la carga de pruebas de la multiplicación de matrices la productividad de cada uno de los nodos era diferente ya el factor principal para su finalización era la capacidad de procesamiento numérico.

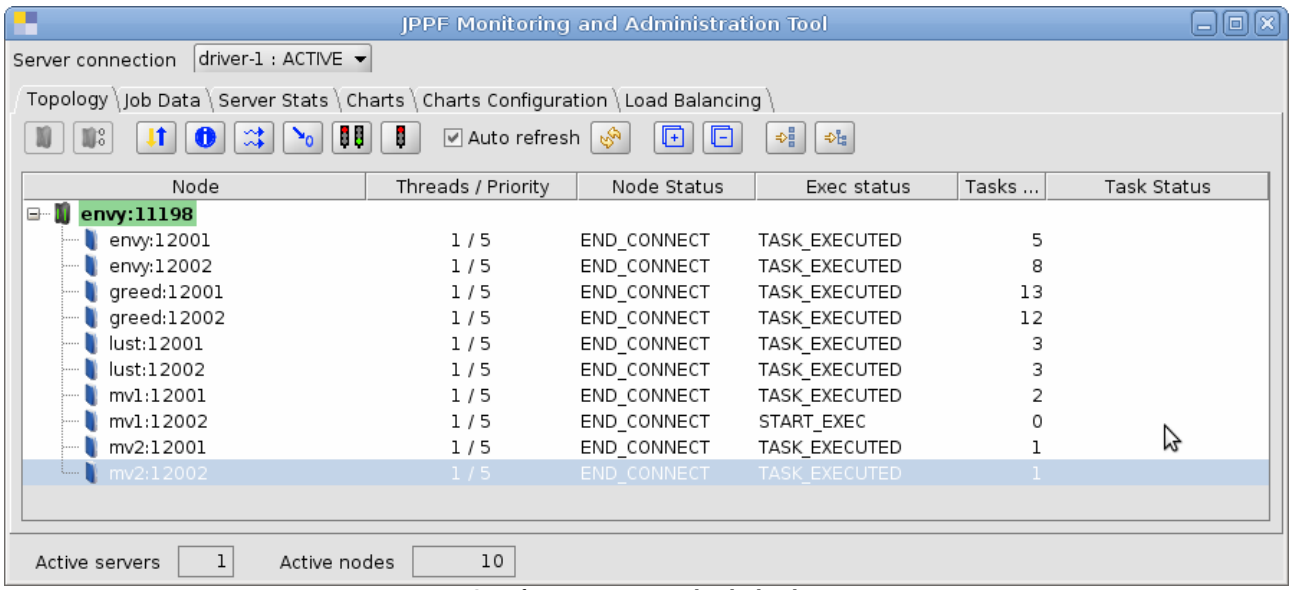


Figura 43 - Infraestructura con nodos duplicados

En la figura anterior se puede observar como la carga entre nodos iguales se ha repartido. Por ejemplo en los nodos Greed que contienen el LogBook del sistema se puede observar que uno de los nodos ha ejecutado 12 trabajos y el otro 13 siendo el total de los trabajos a ejecutar 25. Esta suma coincide en todos los nodos duplicados.

La ejecución sobre 10 expedientes ha sido la siguiente:

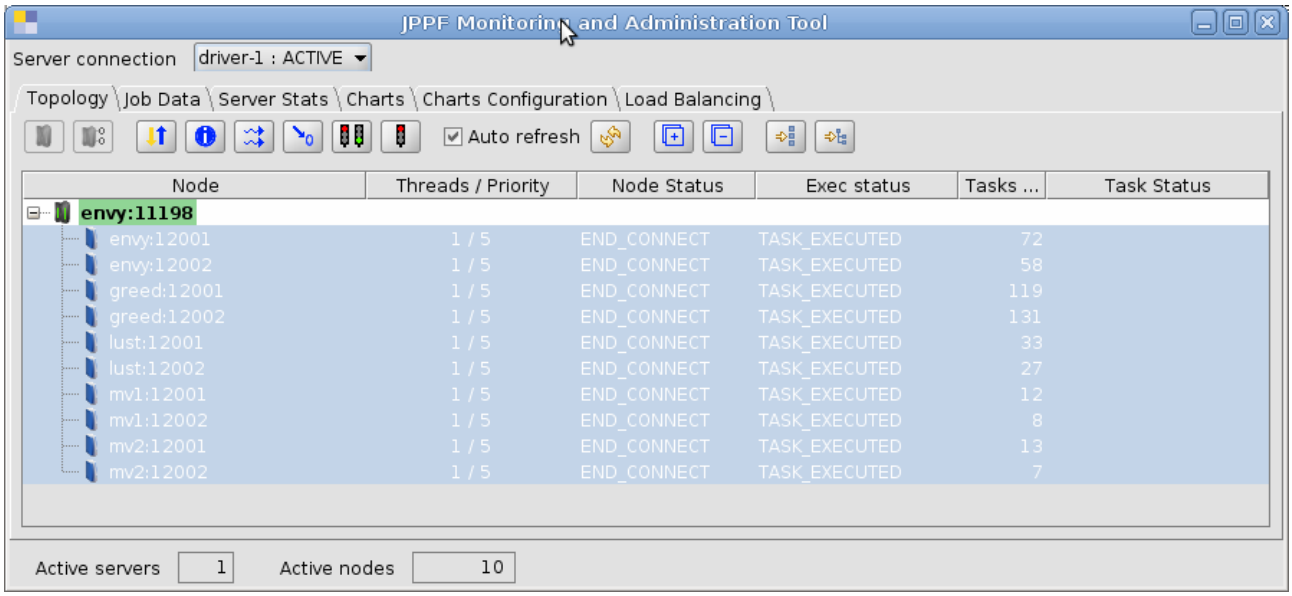


Figura 44 - Resultado de la ejecución de 10 expedientes

Se puede observar que en los nodos greed el total 119+131 suma el total de 250 para la ejecución de 25 trabajos para 10 expedientes. La suma de todos los nodos de los mismos servicios también es correcta.

B2.- El procedimiento de parada de uno de los nodos ha sido correcto y durante el período en el que ha estado inactivo la carga se ha ejecutado por el nodo espejo que posee. Este test se ha realizado con el caso anterior de 10 expedientes.

B3.- La recuperación de los nodos después de pararlos ha funcionado correctamente.

B4 y B5.- No procede

Grupo C.- Pruebas de estrés del sistema (carga).

C1.- No procede.

C2.- Para la realización de las pruebas y evaluación de la ejecución de numerosos trabajos dentro de la infraestructura diseñada se obtendrá el tiempo medio de la ejecución de un expediente.

Total de expedientes concurrentes #nodos x servicio	5 Expedientes Simultáneos	10 Expedientes Simultáneos	15 Expedientes Simultáneos	20 Expedientes Simultáneos
1	269	545	830	1121
2	198	379	579	788
Dif. entre 1 a 2	- 26.3 %	- 30,4 %	- 30,2 %	- 29.7 %

Tabla 12 - Tiempo medio de ejecución en segundos

En las columnas podemos ver el número de trabajos que se ejecutan simultáneamente y en las filas podemos ver el total de nodos que existe para cada uno de los servicios que se deben ejecutar. Como se observa, si se introducen más expedientes en el sistema de forma simultánea el tiempo medio empeora. Esto parece engañoso ya que se espera que los tiempos de ejecución sean menores, mientras existan un número mayor de trabajos. Esta tabla parece que quiere indicar que una ejecución menos paralela produce un resultado mejor.

El problema es otro, en este caso la distribución de los procesos por los nodos no se produce ya que se lanzan todos al mismo tiempo y la ejecución de menos trabajos siempre tarda menos ya que todos los trabajos están esperando ejecutarse en el mismo nodo.

Es más destacable ver la tabla observando el número de nodos por servicio ya que, como estaba previsto, el aumento del número de nodos ha reducido los tiempos de ejecución.

Para observar como se comporta el sistema con la misma carga, no como en la tabla anterior donde la carga del sistema para 5 es cuatro veces inferior que el de 20, realizaremos la ejecución de 20 trabajos y calcularemos el tiempo que tardan en ejecutarse. Estos datos se pueden obtener del logbook del sistema.

Total de expedientes concurrentes #nodos x servicio	5 Expedientes Simultáneos	10 Expedientes Simultáneos	15 Expedientes Simultáneos	20 Expedientes Simultáneos
1	1227	1223	1221	1202
2	831	861	861	869
Dif. entre 1 a 2	-32,2 %	-29,5 %	-29,4 %	-27,7 %

Tabla 13 - Tiempo medio de ejecución por expediente para una carga de 20 (seg.)

Como se puede observar, el aumento de los trabajos ejecutándose simultáneamente no supone un cambio apreciable en el tiempo de ejecución total de los trabajos. Sí que tiene importancia el número total de nodos en el sistema. Sería necesario poder incluir más nodos en hardware independiente para conocer si estos datos son reales o son debidos a las limitaciones del prototipo utilizado.

Por otra parte, se observa una mejora de un 30 % aproximadamente con la duplicación de todos los nodos del sistema, lo mismo que en la primera prueba realizada.

La obtención de los tiempos de proceso se ha realizado utilizando el LogBook y estudiando el fichero que genera.

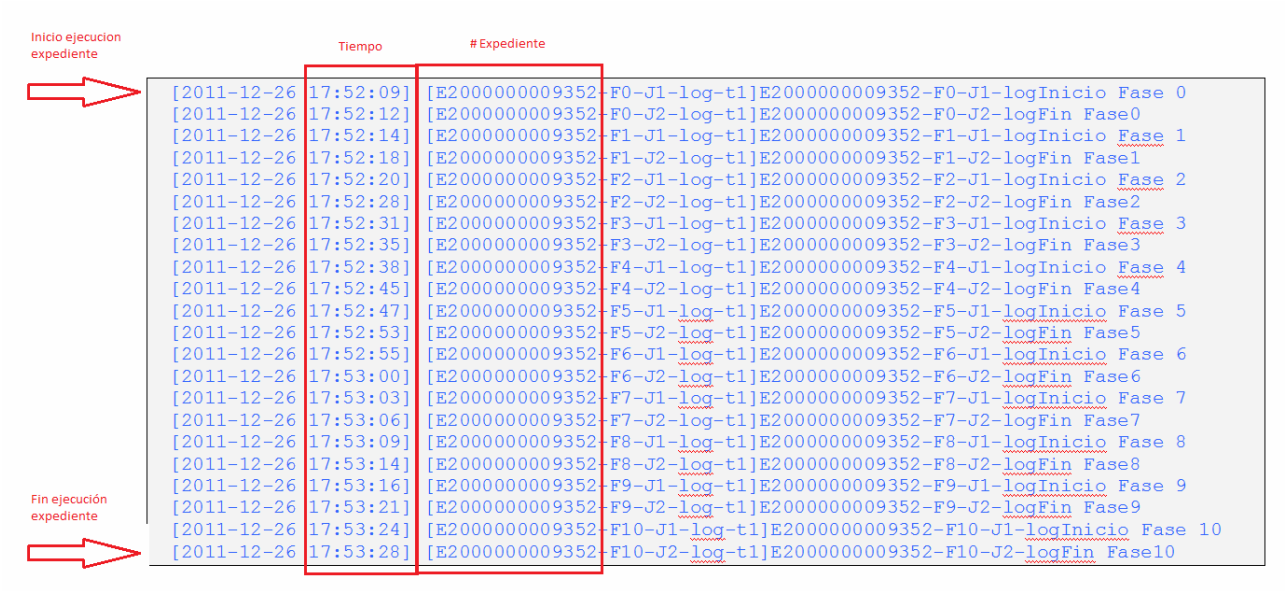


Figura 45 - Captura del logbook

El prototipo se ha comportado correctamente en la ejecución de numerosos trabajos y durante un período de tiempo extenso (varias horas).

C3.- No procede

Grupo D.- Pruebas de escalabilidad del sistema.

D1.- Verificar la mejora de tiempos con el funcionamiento de gran cantidad de nodos. Se utilizarán dos nodos para cada uno de los servicios, de la misma forma que en el caso B1 debiendo aparecer una mejora de los tiempos de ejecución.

Para la verificación de la ejecución en paralelo de los trabajos se ha capturado una pantalla con la ejecución en paralelo de procesos en casi todos los nodos del sistema.

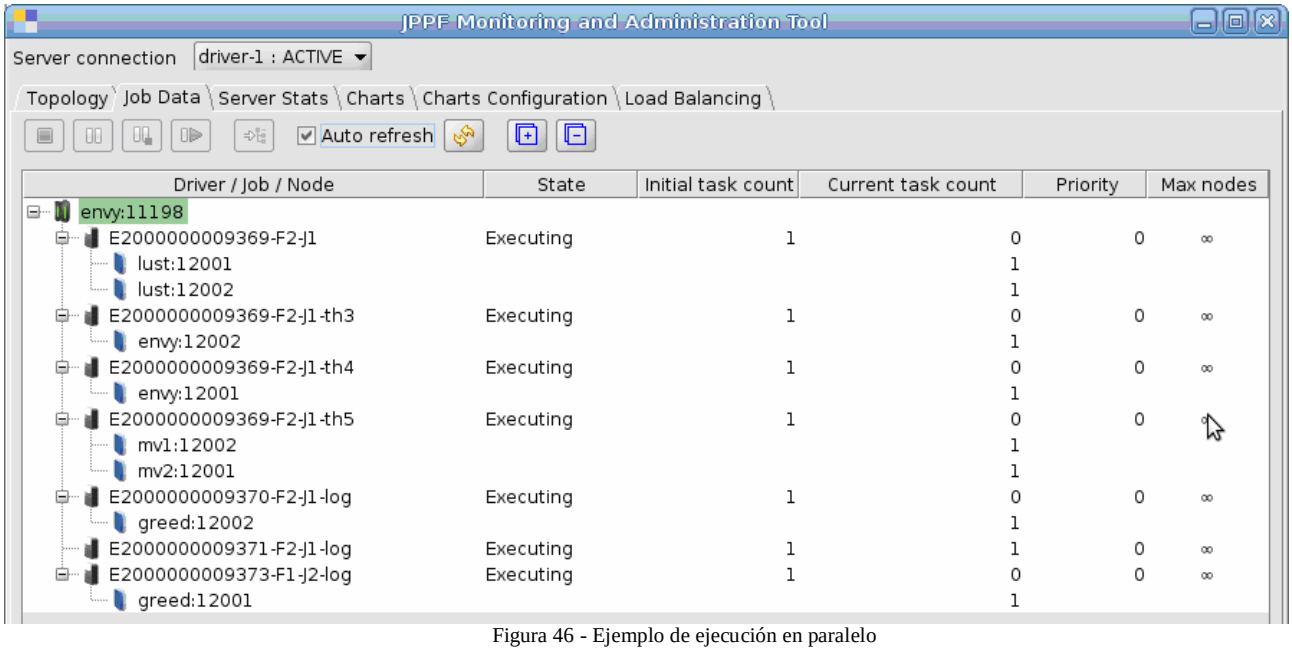


Figura 46 - Ejemplo de ejecución en paralelo

La verificación de mejor ejecución con más nodos se ha realizado en B1 por ello no se repetirá.

D2.- No procede

Grupo E.- Comportamiento del interface gráfico de usuario. No procede.

7.2.3.- Limitaciones encontradas

La realización de las pruebas ha tenido muchas limitaciones que pueden causar la alteración de los datos de las mismas. Las describo a continuación con el fin de poder tenerlas en cuenta:

- Elevada utilización de la CPU. La pruebas realizadas, debido a la estructura de máquinas virtuales, ha dado lugar a tiempos de utilización de CPU muy elevados. Teniendo en cuenta que los trabajos no contienen procesos que produzcan esta carga, esto puede alterar los resultados de la ejecución de pruebas con numerosos nodos y es la principal razón de no seguir la progresión de la utilización de más nodos en las tablas presentadas.
- Número máximo de máquinas utilizables. El número máximo de hardware limita de una forma clara el número de nodos que se pueden utilizar. Ésta es la principal razón de las limitaciones en el estudio.
- Un único thread para la ejecución de los trabajos en los nodos. En cada uno de los nodos se ha permitido solo la ejecución de un único thread de trabajo para facilitar el conteo de los trabajos a realizar. La incorporación de varios thread por nodo puede alterar los resultados de las pruebas realizadas. JPPF permite la utilización de cada nodo por un Tarea (*Task*) si ésta tiene varios Trabajos (*Jobs*) se podrán ejecutar en paralelo por diferentes threads, siempre que sean de la misma tarea. JPPF aconseja que el número de threads ejecutándose en una máquina sea similar al número de cores disponibles para la ejecución de trabajos.

Parte V - Conclusiones y futuros trabajos

8.- Conclusiones y líneas de trabajo futuras

8.1.- Enseñanzas adquiridas

Después de realizar el estudio, la programación y las pruebas del prototipo para la paralelización de los expedientes administrativos se pueden destacar algunas enseñanzas adquiridas y que pueden ser entendidas dentro del contexto de este trabajo.

- Es más modular. La utilización de un entorno grid para la tramitación de los expedientes administrativos permite una modularización e independencia de los componentes que no se puede conocer con la programación común. Cada uno de los módulos puede ser independiente, no solo lógica o funcionalmente, sino también físicamente.
- No se puede realizar sin la abstracción de la orientación a objetos. La complejidad del trabajo a realizar no puede ser realizada si no es mediante el paradigma de programación orientada a objetos. Esto influye en gran manera en los sistemas de gestión actuales en los cuales los objetos se aplican en gran medida a los sistemas Web pero escasamente a los sistemas de gestión.
- Permite la escalabilidad como parte del framework. La escalabilidad de los sistemas, en general, se basa en la utilización de una máquina más grande con mayor capacidad de proceso que se reparte en todos los procesos de una forma más o menos jerárquica. Con la utilización de un GRID la inclusión de potencia adicional resulta más sencilla y mucho más intuitiva.
- Tratamiento correcto de los picos de tareas. Como consecuencia de la escalabilidad de las tareas y/o trámites dentro de un expediente administrativo se puede extraer que la realización de tareas con picos de utilización puede ser tratada de una forma lógica y ordenada.
- Contratación y outsourcing. Podemos justificar la contratación de servicios de cómputo externo que permitan esa utilización en los picos de trabajo. Podemos llegar a utilizar de forma lógica y razonada complementos hardware contratados externamente.
- Disminución de los costes de trabajo. Una vez diseñado y en funcionamiento, la actualización del hardware de todos por uno con mayor potencia puede ser una cuestión trivial. Siempre, eso sí, con la documentación adecuada.
- Aumento de los costes de diseño. El diseño de los procesos se torna más complejo, lo que puede repercutir en un corte más elevado. Debemos tener en cuenta que la mayoría de tiempo de trabajo de un proyecto se lo lleva la fase de análisis y mantenimiento.
- Se obtiene un gran beneficio en el tratamiento de expedientes. Los trámites se realizan en el lugar indicado y en el momento indicado. No existen procesos complejos de transferencia de información y de creación de duplicados parciales dependiendo de las competencias

exigidas. La valoración general es que el tratamiento de los expedientes puede obtener una mejora muy elevada.

- Disminución de Costes Hardware. Debemos tener en cuenta que los entornos Grid están diseñados para funcionar con entornos heterogéneos y por ello la infraestructura tiene un coste inferior para la funcionalidad que proporciona. También se pueden adaptar mejor a las mejoras que existan en el mercado de una forma natural y adecuada.

8.2.- Líneas de trabajo futuras

Durante la realización del trabajo se han dejado fuera numerosas cuestiones que podrían formar parte de nuevos estudios.

- **Seguridad**. Todo proceso de tratamiento de expedientes tiene incluido un tratamiento de datos que por cuestiones legales debe tener un tipo de seguridad asociada. En este trabajo lo único que se ha hecho es mencionar que se puede incluir seguridad al framework elegido, pero no se han tratado en profundidad las necesidades que se poseen. Sin un estudio de seguridad el tratamiento de los expedientes es imposible de implantar.
- **Estudio coste beneficio**. Cuando introducimos un nuevo sistema informático posee un coste económico que debe justificarse generalmente ante terceras personas. Es necesario tener muy claro que el sistema es viable económicamente, pero tal vez lo más importante es la mejora que se obtendría ante el sistema anterior. En nuestro caso deberíamos comparar la arquitectura GRID ante las propuestas de mainframes que se manejan en los sistemas de cómputo de los organismos que tratan los expedientes.
- **Conexión entre GRIDS**. Los entornos de tratamiento de este tipo de información son heterogéneos. El pretender que se pueda utilizar un solo tipo de framework para la realización del tratamiento de expedientes resulta un poco utópico y alejado de la realidad. Por todo ello, sería necesario estudiar de una forma adecuada la interconexión entre sistemas GRID. De la misma forma que existen estudios sobre la interconexión de redes es necesario plantearse la necesidad de intercambiar información, datos, procesos, etc. entre frameworks iguales o diferentes. No es extraño que las tecnologías GRID se estandaricen pero parece que el futuro no es que lo hagan en un solo producto.
- **Escalabilidad**. Se ha evaluado la posibilidad de utilizar JPPF como herramienta para soportar el tratamiento de expedientes, que puede ser una labor de una necesidad de información muy grande. Esto hace que la infraestructura tienda a aumentar de forma gradual con la realización de expedientes más complejos, en base a las nuevas características, o bien el número por el aumento de tipo de expedientes tratados. Todo ello nos lleva a tener que estudiar las posibilidades de crecimiento de una infraestructura ya no de forma teórica sino con el framework que elijamos. En el caso de este trabajo sería JPPF pero cualquiera de los frameworks que se comentaron en el inicio tendría esta necesidad.
- **Desarrollo de aplicaciones con soporte para el tratamiento de expedientes**. El trabajo realizado, limitado especialmente por una cuestión de tiempo, no deja de ser una propuesta de trabajo para la implantación. Es necesario un estudio sobre la capacidad de mejorar de un sistema cuando se realizan cambios arquitectónicos de similar forma que los sistemas C/S fueron paulatinamente sustituidos por los multicapa. El desarrollo de Gestores de Expedientes que permitieran trabajos de diferentes tipos en función de parámetros del propio expediente sería una mejora a la propuesta actual.
- **Nivel de utilización de los servicios**. La propuesta de servicios está diseñada para ser independiente de los expedientes y la entidad que los desee ejecutar. Esto puede dar lugar a sobrecargaa debido a errores o malas utilizaciones de los mismos. Una incorrecta utilización de los servicios puede repercutir en la utilización global de todo el sistema. Sería necesario estudiar los lapsos de tiempo que tienen asignados para no llegar a situaciones de parada total del sistema.

- **Servicios de alto nivel distribuidos entre diferentes máquinas, incluyendo un sistema de ficheros y la atención asíncrona de eventos**. Se han hecho mención a varios servicios que deberían estudiarse para poderlos incluir en sistemas reales. La representación de servicios, que se ha utilizando, permite realizar una abstracción de la infraestructura de la misma forma que un sistema operativo aísla, con sus servicios, de los detalles de implementación. En los dos casos se logra un fin similar, mostrar unos servicios comunes a los usuarios finales.

Parte VI - Apéndices y Bibliografía

9 .-Apéndices

Anexo 1 – Código ejemplo de expediente (XML)

```
<EXPEDIENTE>
 <SYSTEM>
       <SYS_NAME>Gestor_expedientes</SYS_NAME>
       <SYSTEM_VERSION>1.0</SYSTEM_VERSION>
 </SYSTEM>
 <EXP_ID>
       <COD_ID>460012544587</COD_ID>
       <NAME>Grid_Valencia_expediente_tr</NAME>
       <TIPO>TR</TIPO>
       <VERSION>1_0_0</VERSION>
       <ETIME>2011-12-20T09:30:10.5</ETIME>
 </EXP_ID>
       <EXP_STAT>
       <STAT>10</STAT>
       <STAT_TIME>2011-12-20T10:37:11.6</STAT_TIME>
       <LAST_STAT>9</LAST_STAT>
 </EXP_STAT>
 <PARAMETERS>
       <PARAM ord="1">
             <PNAME>Existe_matricula</PNAME>
             <PVALUE>SI</PVALUE>
       </PARAM>
       <PARAM ord="2">
             <PNAME>IRPF_OK</PNAME>
             <PVALUE>SI</PVALUE>
       </PARAM>
       <PARAM ord="3">
             <PNAME>Deudas</PNAME>
             <PVALUE>NO</PVALUE>
       </PARAM>
 </PARAMETERS>
 <FILES>
       <FILE ord="1">
             <SERVICE>general.files</SERVICE>
             <PATH>almacen2</PATH>
             <FNAME>Modelo600_Exp460012544587_digitalizacion.pdf</FNAME>

       </FILE>
       <FILE ord="2">
             <SERVICE>ga1.files</SERVICE>
             <PATH>almacen1</PATH>
             <FNAME>generado10101010.doc</FNAME>

       </FILE>
 </FILES>
</EXPEDIENTE>
```

Anexo 2 – XSD de validación del expediente

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <xs:element name="EXPEDIENTE">
        <xs:complexType>
              <xs:sequence>
                    <xs:element ref="SYSTEM" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="EXP_ID" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="EXP_STAT" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="PARAMETERS" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="FILES" minOccurs="1" maxOccurs="1"/>
              </xs:sequence>
        </xs:complexType>
  </xs:element>

  <xs:element name="SYSTEM">
        <xs:complexType>
              <xs:sequence>
                    <xs:element ref="SYS_NAME" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="SYSTEM_VERSION" minOccurs="1" maxOccurs="1"/>
              </xs:sequence>
        </xs:complexType>
  </xs:element>

  <xs:element name="EXP_ID">
        <xs:complexType>
              <xs:sequence>
                    <xs:element ref="COD_ID" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="NAME" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="TIPO" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="VERSION" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="ETIME" minOccurs="1" maxOccurs="1"/>
              </xs:sequence>
        </xs:complexType>
  </xs:element>

  <xs:element name="EXP_STAT">
        <xs:complexType>
              <xs:sequence>
                    <xs:element ref="STAT" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="STAT_TIME" minOccurs="1" maxOccurs="1"/>
                    <xs:element ref="LAST_STAT" minOccurs="1" maxOccurs="1"/>
              </xs:sequence>
        </xs:complexType>
  </xs:element>

  <xs:element name="PARAMETERS">
        <xs:complexType>
              <xs:sequence>
                    <xs:element ref="PARAM" minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
        </xs:complexType>
  </xs:element>

  <xs:element name="FILES">
        <xs:complexType>
              <xs:sequence>
                    <xs:element ref="FILE" minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
        </xs:complexType>
  </xs:element>

  <xs:element name="SYS_NAME" type="xs:string"/>

  <xs:element name="SYSTEM_VERSION">
        <xs:simpleType>
              <xs:restriction base="xs:string">
                    <xs:enumeration value="1.0"/>
                    <xs:enumeration value="1.1"/>
                    <xs:enumeration value="1.2"/>
              </xs:restriction>
```

```
        </xs:simpleType>
  </xs:element>

  <xs:simpleType name="version">
        <xs:restriction base="xs:string">
               <xs:enumeration value="1.0"></xs:enumeration>
               <xs:enumeration value="1.1"></xs:enumeration>
        </xs:restriction>
  </xs:simpleType>

  <xs:element name="COD_ID" type="xs:string"/>
 <xs:element name="NAME" type="xs:string"/>

  <xs:element name="TIPO">
        <xs:simpleType>
               <xs:restriction base="xs:string">
                      <xs:enumeration value="TR"/>
                      <xs:enumeration value="SUCYDON"/>
                      <xs:enumeration value="PATRI"/>
               </xs:restriction>
        </xs:simpleType>
  </xs:element>

  <xs:element name="VERSION" type="xs:string"/>
  <xs:element name="ETIME" type="xs:dateTime"/>
  <xs:element name="STAT" type="xs:int"/>
  <xs:element name="STAT_TIME" type="xs:dateTime"/>
  <xs:element name="LAST_STAT" type="xs:int"/>

  <xs:element name="PARAM">
        <xs:complexType>
               <xs:sequence>
                      <xs:element ref="PNAME" minOccurs="1" maxOccurs="1"/>
                      <xs:element ref="PVALUE" minOccurs="1" maxOccurs="1"/>
               </xs:sequence>
                      <xs:attribute name="ord" type="xs:positiveInteger"/>
        </xs:complexType>
  </xs:element>

  <xs:element name="PNAME" type="xs:string"/>

  <xs:element name="PVALUE" type="xs:string"/>

  <xs:element name="FILE">
        <xs:complexType>
               <xs:sequence>
                      <xs:element ref="SERVICE" minOccurs="1" maxOccurs="1"/>
                      <xs:element ref="PATH" minOccurs="1" maxOccurs="1"/>
                      <xs:element ref="FNAME" minOccurs="1" maxOccurs="1"/>
               </xs:sequence>
               <xs:attribute name="ord" type="xs:positiveInteger"/>
        </xs:complexType>
  </xs:element>

  <xs:element name="SERVICE" type="xs:string"/>
       <xs:element name="PATH" type="xs:string"/>
  <xs:element name="FNAME" type="xs:string"/>

</xs:schema>
```

Anexo 3 – Ejemplo de código fuente comentado de los nodos desarrollados

El siguiente es el código de uno de los lanzamientos que se realizan para la ejecución en el nodo remoto de los trámites administrativos.

```
package org.uoc.jhh.expedientes.fases;

import java.util.List;
import org.jppf.client.JPPFClient;
import org.jppf.client.JPPFJob;
import org.jppf.node.policy.ExecutionPolicy;
import org.jppf.node.policy.MoreThan;
import org.jppf.server.protocol.JPPFTask;
import org.uoc.jhh.expedientes.tareas.Tarea_verifica_NIF;

/**
 *
 * @author jhh
 */
class Fase_02_expediente_tipo1_Thread1 implements Runnable {
         String codigo_jobs;
         protected JPPFClient jppfClient = null;
         Thread t=null;
         Fase_02_expediente_tipo1_Thread1 (String cj){
                codigo_jobs=cj;
                t=Thread.currentThread();
         }
         @Override
            public void run(){
                try{
                   String show="["+codigo_jobs+"]";
                   jppfClient = new JPPFClient();
                   JPPFJob job1 = new JPPFJob();
                   job1.setId(codigo_jobs);
                   System.out.println("==> Ejecutar Thread “ +
                                      “de fase2 expediente tipo 1 - primera tarea");
                   Tarea_verifica_NIF task1= new Tarea_verifica_NIF("1");
                   task1.setId(codigo_jobs+"-th1");
                   job1.addTask(task1);
                   ExecutionPolicy MoreThanThreadsPolicy = new MoreThan(task1.where_run(), 0);
                   job1.getJobSLA().setExecutionPolicy(MoreThanThreadsPolicy);
                   List<JPPFTask> results1 = jppfClient.submit(job1);
                   String r=null;
                   for (int i=0; i<results1.size(); i++){
                      JPPFTask mTask = results1.get(i);
                      if (mTask.getException() != null) throw mTask.getException();
                            r = (String) mTask.getResult();
                      if ((mTask.getResult() == null )) {
                              System.out.println(show + " ERROR thread gn");
                      }
                      System.out.println (show + "Retorno("+ i +"): " + r);
                   }
                }catch(Exception e){
                    e.printStackTrace();
                    jppfClient.close();
                }
            }
}
```

Anexo 4 – Profundización en JPPF

Con el propósito de explicar ciertas características de JPPF que se utilizan en el proyecto se incluye este anexo con información adicional explicativa del framework.

A4.1.- Puesta en marcha de los módulos (run modes)

La puesta en funcionamiento de los diferentes módulos es bastante sencilla. Todos ellos llevan un fichero .sh (en sistemas Unix) y .bat (para sistemas Windows) para su ejecución. Con la utilidad Ant es posible realizar la compilación en la versión de Java que se tenga disponible dentro del shell de cada una de las máquinas que componen la infraestructura.

En las siguientes capturas se pueden ver la puesta en marcha de un nodo y de un servidor tanto con el script de Linux como con el comando ant.

```
root@envy:/home/joseh/Escritorio/jppf/JPPF-2.5.3-driver# ./startDriver.sh
driver process id: 5864
Class Server initialized - listening on port 11111
Client Server initialized - listening on port 11112
Tasks Server initialized - listening on port 11113
JPPF Driver management initialized
JPPF Driver initialization complete

root@envy:/home/joseh/Escritorio/jppf/JPPF-2.5.3-node# ant run
Unable to locate tools.jar. Expected to find it in /usr/lib/jvm/java-6-
openjdk/lib/tools.jar
Buildfile: /home/joseh/Escritorio/jppf/JPPF-2.5.3-node/build.xml

run:
     [java] node process id: 5970
     [java] Attempting connection to the class server at 172.20.20.60:11111
     [java] Reconnected to the class server
     [java] JPPF Node management initialized
     [java] Attempting connection to the node server at 172.20.20.60:11113
     [java] Reconnected to the node server
     [java] Node sucessfully initialized
```

Figura 4.1 – Inicio del servidor de JPPF

También existe la posibilidad de iniciar los módulos como servicios, tanto en sistemas Windows como Linux.

Una se las formas más peculiares de poner en marcha los nodos es el método “Idle-mode”, que permite realizar operaciones en los nodos solo cuando el host se encuentre sin utilizar. Se considera que está sin utilizar cuando no existe actividad en el teclado o ratón durante un período determinado de tiempo.

A4.2.- Ficheros de configuración

Existen ficheros de configuración diferenciados para cada uno de los nodos, servidores y clientes que se van a ejecutar. En todos los casos el lugar de localización de fichero se puede modificar cuando se pone en marcha el módulo, aunque es normal que se encuentre en el directorio config dentro de la estructura de directorios del módulo, que es el lugar en el que lo sitúan los desarrolladores.

Como ejemplo se muestran los ficheros de configuración del Nodo y del Driver que se han utilizado para alguna de las pruebas realizadas. En [42] existe una breve explicación de cada una de las propiedades

que se muestran. También se muestra en el propio fichero de propiedades una descripción breve de los valores por defecto.

```
jppf.server.host = envy
class.server.port = 11111
app.server.port = 11112
node.server.port = 11113
jppf.management.enabled = true
jppf.management.host = localhost
jppf.management.port = 12001
jppf.management.rmi.port = 13001
jppf.policy.file = config/jppf.policy
jppf.discovery.enabled = true
jppf.discovery.group = 230.0.0.1
jppf.discovery.port = 11111
jppf.discovery.timeout = 5000
reconnect.initial.delay = 1
reconnect.max.time = 5
reconnect.interval = 1
processing.threads = 1
#jppf.object.stream.builder = org.jppf.serialization.GenericObjectStreamBuilder
#jppf.object.stream.builder = org.jppf.serialization.XstreamObjectStreamBuilder
#jppf.object.input.stream.class = java.io.ObjectInputStream
#jppf.object.output.stream.class = java.io.ObjectOutputStream
#jppf.object.input.stream.class = org.jppf.serialization.JPPFObjectInputStream
#jppf.object.output.stream.class = org.jppf.serialization.JPPFObjectOutputStream
jppf.jvm.options = -Xmx128m
#jppf.jvm.options = -server -Xmx128m -
Xrunjdwp:transport=dt_socket,address=localhost:8000,server=y,suspend=n
#jppf.data.transform.class = org.jppf.data.transform.DESCipherTransform
#jppf.idle.mode.enabled = false
#jppf.idle.detector.factory = org.jppf.example.idlesystem.IdleTimeDetectorFactoryImpl
#jppf.idle.timeout = 6000
#jppf.idle.poll.interval = 1000
#jppf.recovery.enabled = true
#jppf.recovery.server.port = 22222
#jppf.recovery.max.retries = 2
#jppf.recovery.read.timeout = 60000
#jppf.discovery.ipv4.include = 192.168.1.
#jppf.discovery.ipv4.exclude = 192.168.1.-9; 192.168.1.100-
#jppf.discovery.ipv6.include = 1080:0:0:0:8:800:200C-20FF:-
#jppf.discovery.ipv6.exclude = 1080:0:0:0:8:800:200C-20FF:0C00-0EFF
```

Figura 4.2 – Fichero jppf-config.properties (para nodo)

```
jppf.server.host = 172.20.20.60
#class.server.port = 11111
#app.server.port = 11112
#node.server.port = 11113
#jppf.management.enabled = true
#jppf.management.host = localhost
#jppf.management.port = 11198
#jppf.management.rmi.port = 12198
#jppf.discovery.enabled = true
#jppf.discovery.group = 230.0.0.1
#jppf.discovery.port = 11111
#jppf.discovery.broadcast.interval = 1000
jppf.peer.discovery.enabled = true
jppf.load.balancing.algorithm = proportional
jppf.load.balancing.strategy =  autotuned
strategy.manual.size = 1
strategy.autotuned.size = 5
strategy.autotuned.minSamplesToAnalyse = 100
strategy.autotuned.minSamplesToCheckConvergence = 50
strategy.autotuned.maxDeviation = 0.2
strategy.autotuned.maxGuessToStable = 50
strategy.autotuned.sizeRatioDeviation = 1.5
strategy.autotuned.decreaseRatio = 0.2
strategy.proportional.size = 5
strategy.proportional.performanceCacheSize = 300
strategy.proportional.proportionalityFactor = 1
strategy.rl.performanceCacheSize = 1000
strategy.rl.performanceVariationThreshold = 0.001
strategy.rl.maxActionRange = 10
strategy.test.size = 1
strategy.test.minSamplesToAnalyse = 100
strategy.test.minSamplesToCheckConvergence = 50
strategy.test.maxDeviation = 0.2
strategy.test.maxGuessToStable = 50
```

```
strategy.test.sizeRatioDeviation = 1.5
strategy.test.decreaseRatio = 0.2
strategy.test.performanceCacheSize = 1000
strategy.test.proportionalityFactor = 1
strategy.test.increaseRate = 0.03
strategy.test.rateOfChange = 0.9
strategy.test.discountFactor = 0.2
strategy.test.performanceVariationThreshold = 0.001
strategy.test.maxActionRange = 10
jppf.jvm.options = -Xmx256m
#jppf.jvm.options = -server -Xmx256m -
Xrunjdwp:transport=dt_socket,address=localhost:8000,server=y,suspend=n
#jppf.data.transform.class = org.jppf.example.dataencryption.SecureKeyCipherTransform
#jppf.object.stream.builder = org.jppf.serialization.GenericObjectStreamBuilder
#jppf.object.stream.builder = org.jppf.serialization.XstreamObjectStreamBuilder
#jppf.object.input.stream.class = java.io.ObjectInputStream
#jppf.object.output.stream.class = java.io.ObjectOutputStream
#jppf.object.input.stream.class = org.jppf.serialization.JPPFObjectInputStream
#jppf.object.output.stream.class = org.jppf.serialization.JPPFObjectOutputStream
#jppf.data.transform.class = org.jppf.example.dataencryption.SecureKeyCipherTransform
#jppf.local.node.enabled = false
#jppf.idle.mode.enabled = false
#jppf.idle.detector.factory = org.jppf.example.idlesystem.IdleTimeDetectorFactoryImpl
#jppf.idle.timeout = 6000
#jppf.idle.poll.interval = 1000
#jppf.recovery.enabled = false
#jppf.recovery.max.retries = 3
#jppf.recovery.read.timeout = 6000
#jppf.recovery.server.port = 22222
#jppf.recovery.reaper.run.interval = 60000
#jppf.recovery.reaper.pool.size = 8
```

Figura 4.3 – Fichero jppf-config.properties (para Driver)

A4.3.- Administración y Monitorización

Las tareas de administración y monitorización se realizan mediante la consola de administración que se distribuye como un módulo de la aplicación. La herramienta está basada en el estándar JMX (*Java Management eXtensions*) [40]. La monitorización muestra información de los Servidores, Nodos, Tareas y Jobs que se están en funcionamiento en cada momento dentro del entorno. Muestra valores estadísticos sobre las colas de ejecución de los trabajos, los tiempos medios de ejecución, etc.

De la misma forma que se ponen en funcionamiento los nodos y los servidores, se puede poner en funcionamiento el módulo de monitorización.

```
joseh@envy:~/Escritorio/jppf/JPPF-2.5.3-admin-ui$ ./startConsole.sh
[client: driver-1] Attempting connection to the class server at envy:11111
[client: driver-1] Reconnected to the class server
[client: driver-1] Attempting connection to the JPPF task server at
envy:11112
[client: driver-1] Reconnected to the JPPF task server
```

Figura 4.4 – Puesta en marcha de la consola de monitorización

Esta ejecución muestra en la consola gráfica la información que se observa en la siguiente figura y que es actualiza periódicamente. Se pueden realizar operaciones básicas como reinicio de servidores o la parada/reinicio de nodos o drivers (servidores).

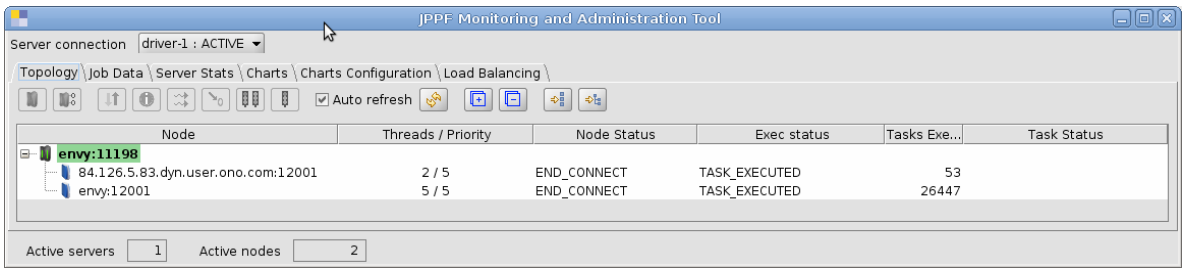


Figura 4.5 – Captura de la aplicación de monitorización

El método para la obtención de información de los nodos es la utilización de la tecnología JMX (*Java Management eXtensions*) [40]. Dentro de los Nodos tenemos dos MBean:

- MBean name: "**org.jppf:name=admin,type=node**". Que se encuentra en la constante JPPFAdminMBean.NODE_ADMIN_NAME. La tarea que realiza este MBean es la de monitorización y administración a nivel de nodo.
- MBean name: "**org.jppf:name=task.monitor,type=node**". Que se encuentra en JPPFNodeTaskMonitorMBean.TASK_MONITOR_MBEAN_NAME como constante. Este MBean monitoriza la actividad de las tareas del nodo.

A nivel de servidor tenemos:

- MBean name: "**org.jppf:name=admin.,type=driver**". Que se encuentra en la constante DRIVER_MBEAN_NAME.
- MBean name: "**org.jppf:name=jobManagement,type=driver**".

El modelo de comunicación utilizado por JPPF es RMI (*Remote Method Invocation*). RMI es el método más sencillo para la comunicación entre programas Java ya que está integrado en la propia tecnología. Tanto los nodos como los drivers disponen de su propio RMIRegistry que permite informar de los servicios disponibles. Los puertos utilizados para la comunicación entre los diferentes componentes se definen en el fichero de configuración del elemento de monitorización.

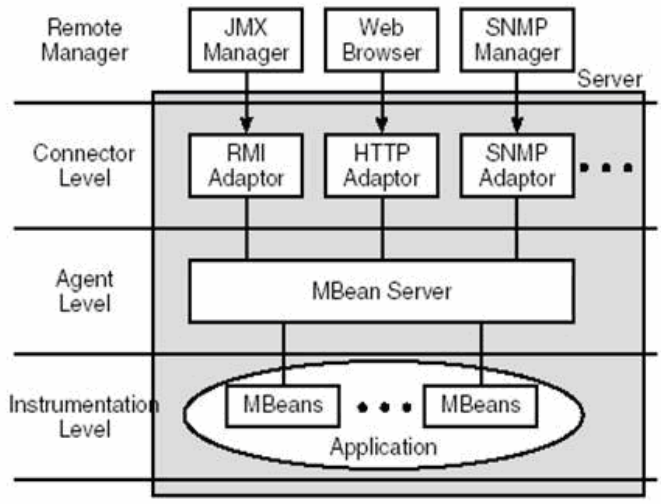


Figura 4.6 – Arquitectura JMX

En la consola podemos acceder a los datos del estado de los trabajos. Así consideraremos que los trabajos pueden hallarse en uno de los siguientes estados:

- Finalizado. Cuando se ha terminado el trabajo.
- Suspendido. Está en el sistema en espera de algún evento.
- En Ejecución. Tiene asignado uno o varios nodos para su ejecución.

A4.4.- Balanceo de carga

El balanceo de carga se refiere a la forma en que los jobs son divididos en sub-trabajos y que estos sub-trabajos son distribuidos en los nodos para su ejecución en paralelo. Cada uno de los sub-jobs contiene un subgrupo de tareas, diferentes del trabajo original.

La distribución de las tareas a los nodos la realiza el Driver o servidor. En estos momentos este factor es el que tiene mayor importancia en la productividad de la aplicación. Un conjunto de tareas enviadas a un nodo se le denomina “bundle” (manojo) y la tarea del algoritmo del balanceador de carga o distribuidor de tareas es optimizar el rendimiento, remitiendo las tareas a los diferentes nodos. En definitiva debe calcular el número adecuado de jobs para cada nodo.

Se puede adjuntar información adicional (metadata) a los trabajos para describir sus características tanto del job como de las tareas. Esta información se puede utilizar para personalizar el balanceo de carga y realizar un algoritmo basado en el conocimiento del tipo de carga que se va a ejecutar.

Se permite la realización de balanceadores de carga personalizados que se pueden poner en funcionamiento de una forma rápida y sencilla.

A4.5.- Intercambio de información entre los componentes

El intercambio de información entre componentes se realiza mediante mensajes.

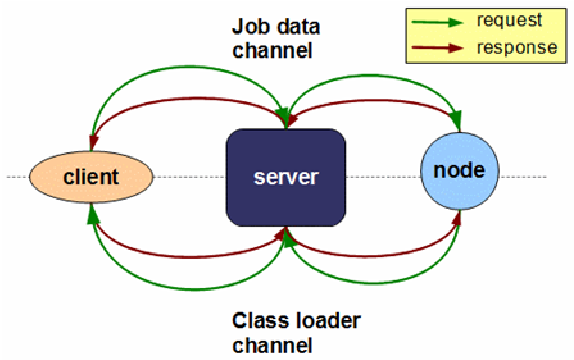


Figura 4.7 – Esquema de intercambio de información. Fuente: [42]

La comunicación entre elementos posee tres reglas básicas para comprenderlo:

- Los nodos solo pueden ejecutar un trabajo (*job*) al mismo tiempo. El trabajo puede estar compuesto por varias tareas que pueden ejecutarse en paralelo.
- Una conexión cliente-servidor solo puede gestionar un *job* pero un cliente puede conectarse varias veces a un servidor o múltiples veces a varios servidores.
- Una conexión servidor-servidor solo puede procesar una aplicación al mismo tiempo. Un servidor se conecta a otro de la misma forma que lo hace un nodo.

Los mensajes tienen siempre la misma organización. Uno o varios bloques de información (bloques de bits) precedidos cada uno de ellos por un bloque con el tamaño. Cada uno de los bloques representa un objeto serializado.

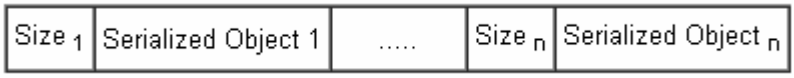


Figura 4.8 – Modelo tipo de mensaje de intercambio. Fuente [42]

Para el canal de información de trabajo, la petición de ejecución de un trabajo (*job data request*), se componen los siguientes elementos:

- Cabecera. Con información sobre el trabajo incluyendo el número de tareas, el SLA, los metadatos e información adicional requerida internamente por JPPF.
- *Data Provider*. Es un contenedor de sólo lectura para la información compartida para todas las tareas.
- Las tareas.

El mensaje queda representado de la siguiente forma:

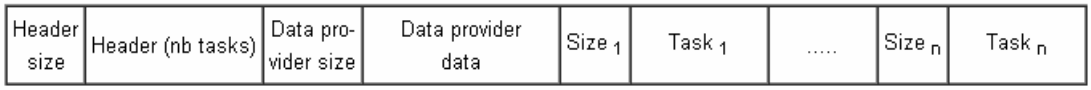


Figura 4.9 – Mensaje de envío de información. Fuente [42]

La respuesta se devuelve en un formato muy similar

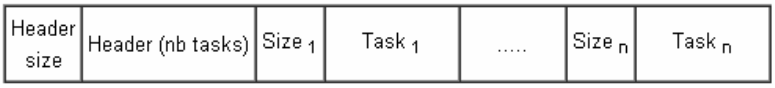


Figura 4.10 – Mensaje de retorno de información. Fuente [42].

Para el canal correspondiente al cargador de clases, se utiliza el mismo formato de mensaje tanto en la solicitud como en la respuesta, compuesto por el objeto serializado con el siguiente formato:

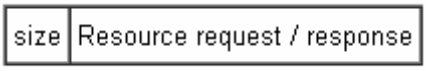


Figura 4.11 – Mensaje de clases. Fuente [42]

A4.6.- SLA (Service Level Agreement)

El SLA (*Service Level Agreement*) o Acuerdo de Nivel de Servicio se define como los términos y condiciones con que los trabajos serán procesados por el servidor. Con este concepto, la herramienta en estudio permite especificar las siguientes características:

- La prioridad del trabajo.
- Número máximo de nodos en el que se puede ejecutar.
- La política de ejecución del nodo, definiéndose las características de los nodos en los que se puede ejecutar.
- El momento de inicio y fin para comenzar un trabajo.
- La suspensión de los trabajos.

El Framework define una política de ejecución como un objeto que determina qué conjunto particular de tareas JPPF pueden ejecutarse en un nodo. Esto se consigue aplicando unas reglas a unas propiedades que posee el nodo.

Las propiedades que se pueden asignar se agrupan en las siguientes categorías:

- Propiedades del sistema.
- Variables del sistema.
- Listas de direcciones IPV4 o IPv6.
- Información de la máquina de ejecución (head memory, número de procesadores disponibles, etc.).
- Espacio en disco para la ejecución.

Las forma de aplicar las SLA es introduciendo estas como políticas de ejecución de los trabajos, indicando al trabajo las características que debe cumplir. Se puede aplicar de dos maneras:

- Por extensión de la clase *CustomPolicy.*
- Por la definición de un fichero de configuración XML en el classpath del sistema o del cliente.

Como ejemplo de las SLA de JPPF podemos observar el siguiente fichero de configuración

```
<ExecutionPolicy>
   <!-- define a policy that requires both rules to be satisfied -->
   <AND>
     <AtLeast>
       <Property>processing.threads</Property>
       <Value>2</Value>
     </AtLeast></nowiki>
     <Contains ignoreCase="true">
       <Property>ipv4.addresses</Property>
       <Value>gva.es</Value>
     </Contains>
   </AND>
 </ExecutionPolicy>
```

Donde se indican dos condiciones. La primera, que el nodo tenga dos threads para el procesamiento (parece una medida que tenga cierta potencia de cálculo) y por otro lado que la IP pertenezca al dominio gva.es con lo que estamos restringiendo el grupo de nodos.

La definición que realiza JPPF del SLA corresponde más con el SLS (*Service Level Specification*) ya que en el SLA se suele indicar más información sobre los servicios, medidas, resolución de problemas, garantías, recuperación ante desastres, finalización de los trabajos, etc. Es cierto que existen opciones cuantitativas pero no opciones cualitativas que también son una tarea de debiera tener en cuenta un SLA. En SLS se describen los parámetros y cantidades que deben cumplirse para conseguir el SLA indicado [12] por lo tanto, el nivel de descripción de JPPF estaría más cercano a este último.

A4.7.- TCP Multiplexer

El TCP Multiplexer tiene la función de realizar la tunelización de los intercambios de información entre el nodo y el Driver. En el lado del Server se define un fichero donde se especifica el puerto por el que se recibirán todos los datos de forma tunelizada.

El esquema del sistema junto al multiplexer es el siguiente:

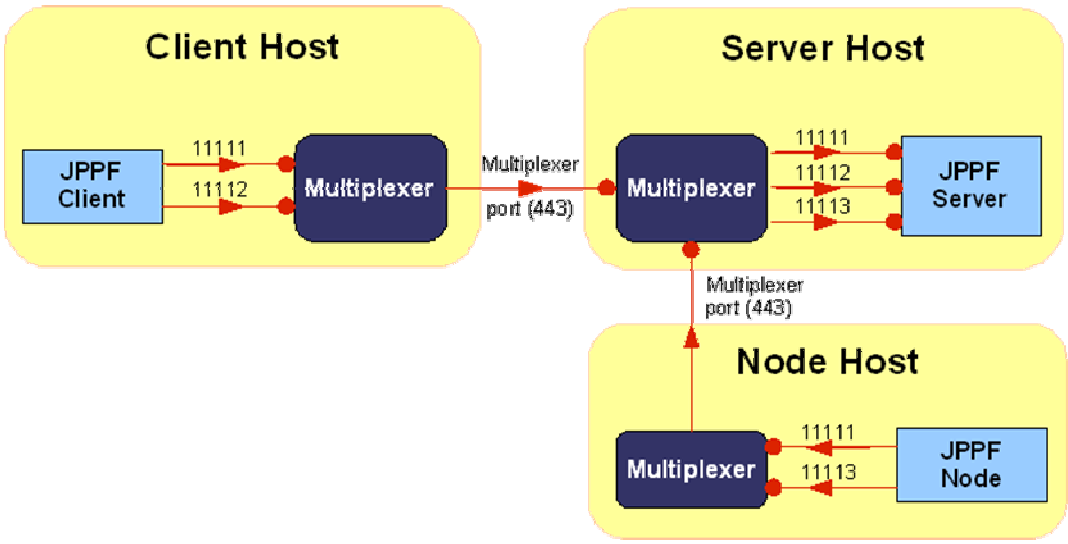


Figura 4.12 – Esquema de posible topología con la utilización del multiplexer. Fuente [42]

El fichero de configuración para el cliente:

```
remote.multiplexers  = m1
remote.multiplexer.m1 = hostenvy:443
bound.ports = 11111 11112 11113
mapping.11111 = m1
mapping.11112 = m1
mapping.11113 = m1
```

Figura 4.13 – Fichero configuración multiplexer (Nodo)

El fichero de configuración para el Driver o Servidor:

```
multiplexer.ports = 443
```

Figura 4.14 – Fichero configuración multiplexer (Driver)

La principal idea de JPPF es realizar una redirección de los puertos necesarios para tunelizarlos a través de Internet y/o otras redes y de esa forma poder realizar una infraestructura con todos los nodos/drivers. Para realizar esta función existen varias opciones en el mercado, muy cercanas a la tecnología de las redes. Para solucionar el problema, JPPF proporciona el multiplexor que es un programa java que realiza la tunelización de los puertos, indicado de una forma transparente.

Los intentos para hacer funcionar el multiplexor a través de Internet no han funcionado aunque se pueden utilizar otras opciones para la tunelización. Para las verificaciones del funcionamiento remoto se ha usado *putty* para redireccionar los puertos necesarios hasta el servidor adecuado.

A4.8.- Conector J2EE

Los adaptadores de recursos son componentes software que permiten a una aplicación J2EE acceder e interactuar con el controlador de recursos subyacentes. Los adaptadores de recursos son específicos de cada recurso.

El *JPPF Resource Adapter* es un componente compatible con JCA (*J2EE Connector Architecture)* [43] que permite a las aplicaciones J2EE el envío de *Jobs* al framework.

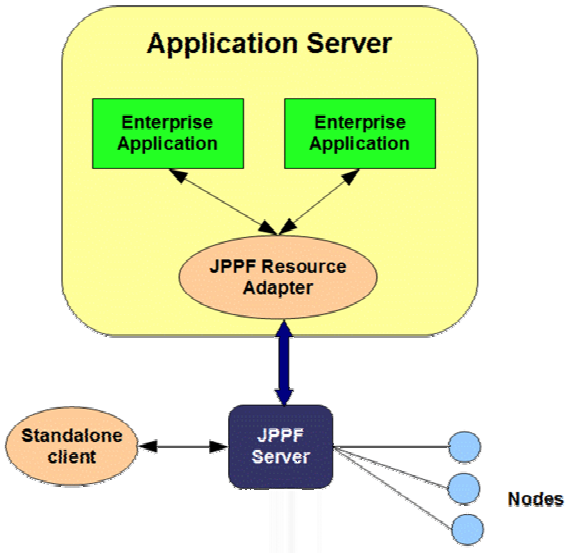


Figura 4.15 – Conector J2EE. Fuente: [42]

A4.9.- Otros temas de interés

Además de las características indicadas en los aparatados anteriores, JPPF posee algunas extensiones que pueden ser utilizadas para ampliar su funcionamiento.

Una de las posibilidades que posee es la incorporación de MBean personalizables para ampliar las posibilidades de monitorización del sistema. Son insertables tanto en los nodos como en el Driver, lo que hace el sistema más transparente a los posibles estudios a realizar.

Otras posibilidad es la puesta en marcha de clases en el arranque del Driver, lo que permite la inicialización de recursos como bases de datos, procesos de autenticación, etc.

Aunque no es el objeto de este trabajo, se debe tener en cuenta que el envío de los objetos serializados se hace sin ningún tipo de encriptación ni seguridad. En el caso que se considere este riesgo se pueden utilizar un mecanismo de encriptación DES o cualquier otro que se considere adecuado. Existe un ejemplo de implementación en los ejemplos que se proporcionan.

La vía de intercambio de los objetos entre los elementos de la infraestructura es por medio de la serialización. Este tipo de conversión hace la transferencia de información a los nodos más fácil y eficiente. Se pueden implantar procedimientos alternativos a los estándares de la aplicación para la serialización. Además de la serialización estándar, JPPF proporciona un serialización basada en XStream [67] que puede ser configurada de forma sencilla.

A4.9.1.- JPPF samples pack

JPPF incorpora en su distribución ejemplos para realización de diferentes tareas para su ejecución en los entornos de desarrollo. Estos ejemplos pueden ser utilizados como herramientas de test para evaluar el correcto funcionamiento de una infraestructura diseñada o para conocer la correcta implementación de tareas en ámbitos determinados de la computación.

Uno de los ejemplos que es utilizable es la implementación de un servidor de FTP.

Estas son algunas muestras de las pruebas que se dispone en el paquete de ejemplo y que han sido utilizadas para las pruebas del prototipo.

- Multiplicación de matrices (MatrixMultiplication). Este código realiza la multiplicación de dos matrices cuadradas densas (una matriz poco densa es aquella que tiene muchos ceros) del tamaño indicado en el fichero de configuración. La operación se realiza en paralelo ejecutando en cada una de las tareas la multiplicación de una columna de la primera matriz por la segunda matriz. Este proceso se puede realizar numerosas veces para permitir el envío a varios nodos.
- Problema de los n-cuerpos. El problema consiste en determinar la posición de n cuerpos dependiendo de las interacciones con los demás. En este caso se simula el movimiento de anti-protones dentro de un campo magnético. Este problema hace uso de una librería paralela de java (*Parallel Java Library*).
- Generación de Fractales. Se generan fractales Maldelbrot. Por la propia definición la generación de fractales es una tarea infinita que puede, dependiendo de la complejidad, llevar a la utilización de un tiempo considerable de cálculo.

Como vemos, se han seleccionado problemas que requieren una gran cantidad de potencia de cálculo para su solución y que será dependiente del número de nodos existentes para su ejecución.

A4.9.2.- Configuración y puesta en marcha

Antes de comenzar la ejecución de pruebas queda pendiente explicar la instalación y configuración realizadas.

La instalación, tanto de los nodos como de los Drivers dentro de la misma LAN, es una tarea rápida, siempre que se cumplan las condiciones de instalación y funcionamiento que hemos comentado en puntos anteriores. Lo podemos resumir en los siguientes pasos:

- Descarga de la Web de JPPF del módulo a utilizar.
- Compilar el módulo con la herramienta ANT (*ant compile*).
- Puesta en marcha con el shell de ejecución.

Es conveniente la puesta en marcha de un primer Driver y después ir añadiendo los nodos de los diferentes equipos. En el momento de la puesta en marcha son los propios nodos los que buscan en la red mediante mensajes de broadcast al Driver que existe y se conectan automáticamente. Es posible que este sea un comportamiento no deseado que se puede prevenir mediante la indicación de la IP del Driver al que se tiene que conectar.

A4.9.3 - Apache Ant

Hemos comentado que es necesaria la compilación con ANT de los ficheros fuente. Apache Ant es una herramienta usada en programación para la realización de tareas mecánicas y repetitivas, normalmente durante la fase de compilación y construcción (build). Es similar a Make pero desarrollado en lenguaje Java y requiere la plataforma Java.

La herramienta, realizada en el lenguaje de programación Java, tiene la ventaja de no depender de las órdenes del shell de cada sistema operativo, sino que se basa en archivos de configuración XML y clases Java para la realización de las distintas tareas, siendo idónea como solución multiplataforma.

ANT fue creado por James Duncan Davidson mientras realizaba la transformación de un proyecto de Sun Microsystems en *open-source* (concretamente la implementación de Servlets y JSP de Sun que luego se llamaría Jakarta Tomcat). En un entorno cerrado, Make funcionaba correctamente bajo plataforma Solaris, pero para el entorno de *open-source*, donde no era posible determinar la plataforma bajo la que se iba a compilar, era necesaria otra forma de trabajar. Así nació Ant como un simple intérprete que cogía un archivo XML para compilar Tomcat independientemente de la plataforma sobre la que operaba. A partir de este punto la herramienta fue adoptando nuevas funcionalidades y actualmente es un estándar en el mundo Java.

Anexo 5 – SQL para la generación de base de datos

```
PRAGMA foreign_keys = ON;

CREATE SCHEMA jppf AUTHORIZATION dba;

drop table jppf.espera;
drop table jppf.log_expedientes;
drop table jppf.expedientes;
drop table jppf.tipo_expedientes;

create table jppf.tipo_expedientes(
         code integer primary key not null,
         name char(100) not null,
         des char(100) not null,
         class char(100) not null
         );

create table jppf.expedientes (
   code  char(230) primary key not null,
   tipo integer not null,
   estatus integer not null,
   waiting boolean not null default false,
   fase integer not null default 0,
   foreign key (tipo) references jppf.tipo_expedientes(code));

create index expediente_tipo_index on expedientes (tipo);

create table log_expedientes(
   exp_code char(230) not null,
   orden integer not null,
   fecha_hora test(30) not null,
   des char (500) not null,
   primary key (exp_code,orden)
);

create index log_expedientes_index on jppf.log_expedientes(fecha_hora);

create table jppf.espera(
   codigo_expediente char(230) not null,
   codigo_espera char(100) not null,
   en_espera integer default 1,
   primary key (codigo_expediente, codigo_espera),
   foreign key (codigo_expediente) references jppf.expedientes (code)
);

create index espera_index1 on jppf.espera(codigo_expediente);
create index espera_index2 on jppf.espera(codigo_espera);

CREATE SEQUENCE jppf.expediente_sec AS BIGINT START WITH 1000000000000;

insert into jppf.tipo_expedientes values(1, "sucesiones",
       "un ejemplo de sucesiones","expediente_sucesiones");
insert into jppf.tipo_expedientes values(2,
       "transmisiones","un ejemplo de transmisiones","expediente_transmisiones");
insert into jppf.tipo_expedientes values(3,
       "actosjuridicos","un ejemplo de ajd","expediente_ajd");
insert into jppf.tipo_expedientes values(4,
       "becasMEC","un ejemplo de beca","expediente_beca");
insert into jppf.tipo_expedientes values(5,
       "Estudio Personal","un ejemplo de estudio","expediente_estudio");

insert into jppf.expedientes values ("1000000000001",1,0,"false",1);
insert into jppf.expedientes values ("1000000000002",2,0,"false",1);
insert into jppf.expedientes values ("1000000000003",3,0,"false",1);
insert into jppf.expedientes values ("1000000000004",4,0,"false",1);
insert into jppf.expedientes values ("1000000000005",5,0,"false",1);
insert into jppf.expedientes values ("1000000000006",1,0,"false",1);

select "fin";
```

Anexo 6 – Configuración agregada a los Ficheros de configuración de los nodos en las pruebas realizadas

Esta es la parte que se ha incluido en los ficheros de configuración (jppf-node.properties) de cada uno de los nodos con el fin de proporcionar los servicios indicados.

Nodo Envy

…
ga1.notificar_interesado=1
ga1.es_correcto.solicita=1
ga1.es_correcto.responde=1
ga1.se_acepta.solicita=1
ga1.se_acepta.responde=1
ga1.verifica_importe_beca=1
ga1.verifica_matricula=1
ga1.verifica_cc=1
ga1.verifica_xml=1
ga1.inicio_expediente=1
ga1.obtener_datos_academicos=1
ga1.solicita_pago=1
ga1.almacenar_expediente=1
ga1.fin_expediente=1
ga1.solicitar_datos.solicita=1
ga1.solicitar_datos.responde=1
ga1.firma_documento.solicita=1
ga1.firma_documento.responde=1

Nodo Greed

…
ga2.verifica_peticiones=1
ga2.obtener_datos_academicos=1
ga2.becas_otra_comunidad=1

general.logbook=1

Nodo Lust

…
gn.verifica_nif=1
gn.verifica_dir=1
gn.verifica_deuda=1
gn.verifica_irpf=1
gn.obtener_datos_academicos=1
gn.solicitar_pnb=1

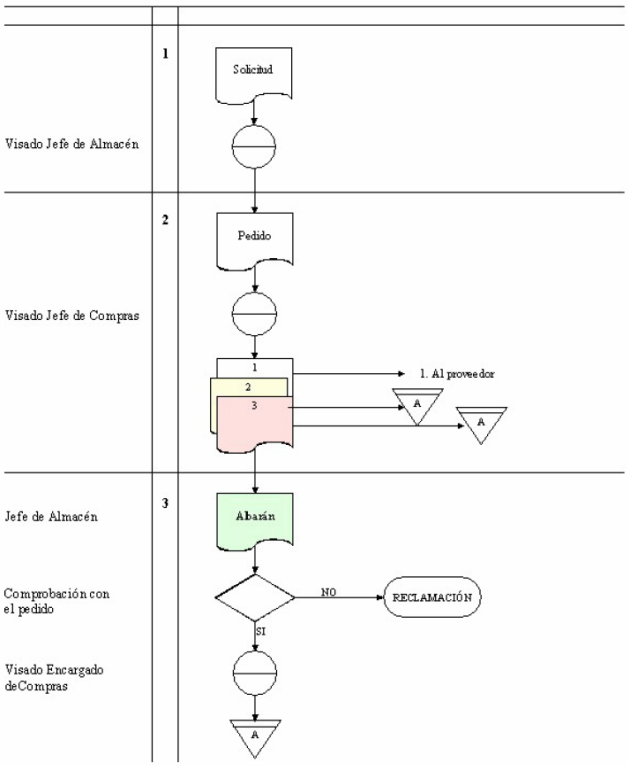
Nodo Mv1

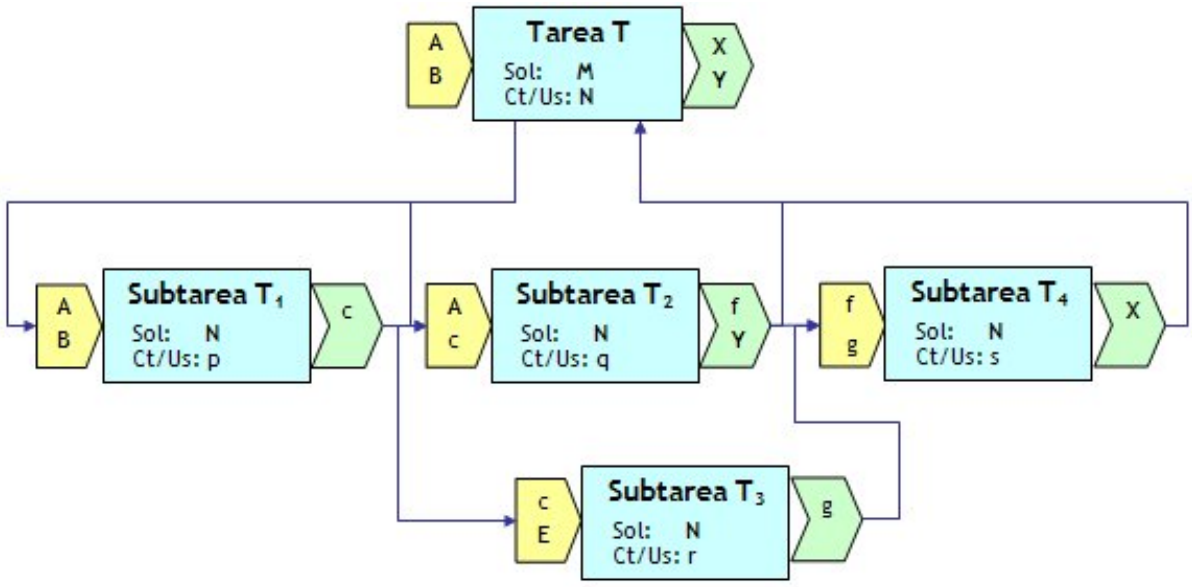
…
ayto1.verifica_peticiones=1
ayto1.becas_ayto=1

Nodo Mv2

…
ayto2.verifica_peticiones =1
ayto2.becas_ayto=1

Anexo 7 – Ejemplos de representación de los diagramas de flujo





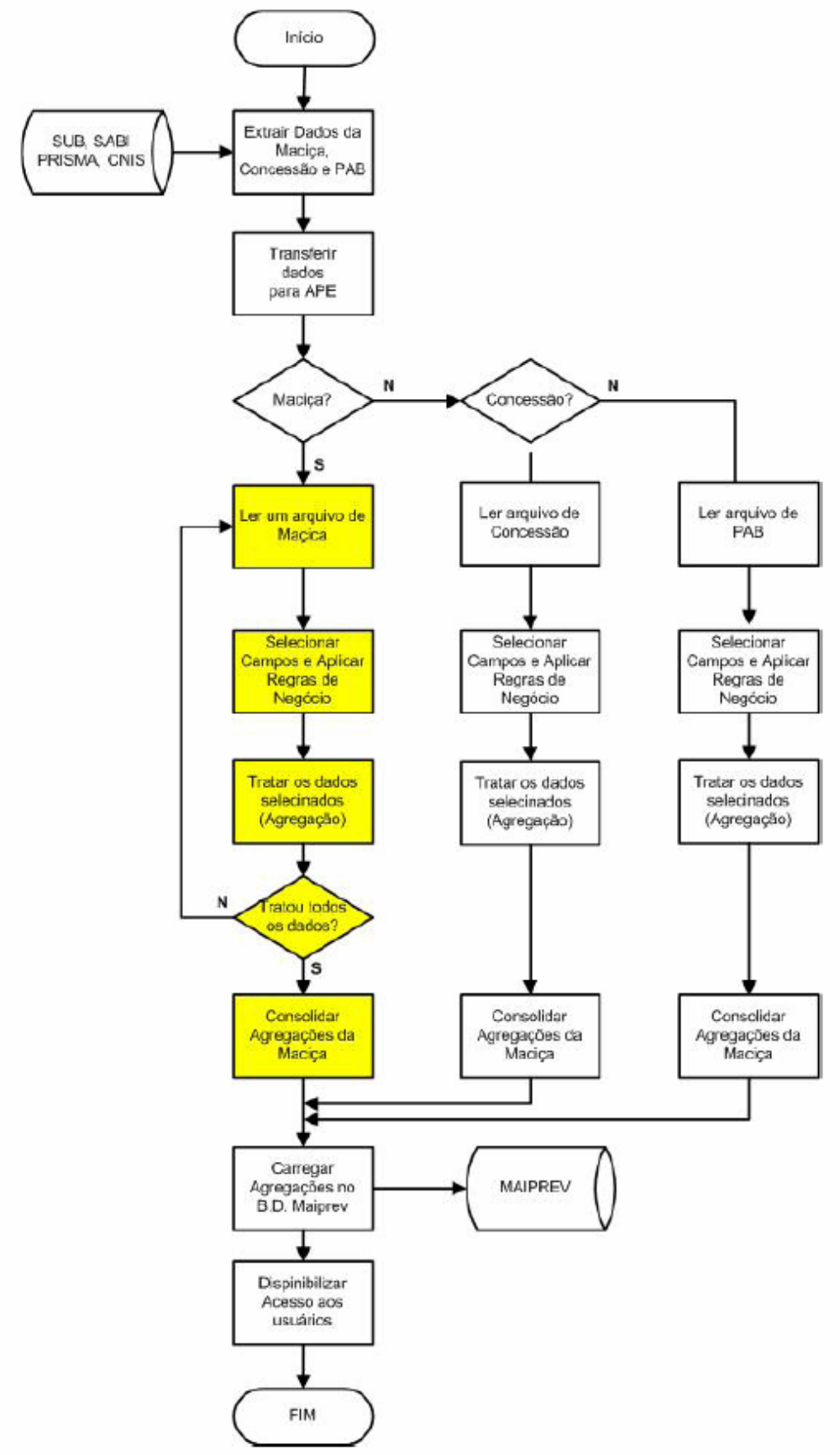


Figura 9.1 - Flujogramas en el tratamiento de trabajos, expedientes, y flujos de aprovisionamiento. Fuente: [58] y otros

10 .- Bibliografía

[01] Acisclo, Enrique; Mendez, Cesar: "**Sistema Integral de Gestión de Ingresos para las Administraciones Públicas**". Web: http://administracionelectronica.gob.es/recursos/pae_020001060.pdf
[02] "**Alfresco Home Page**". Web: http://www.alfresco.com/
[03] "**Alphabetical list of institutions that are using Hadoop**". Web: http://wiki.apache.org/hadoop/PoweredBy
[04] Amdahl, Gene: "**Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities**". AFIPS Conference Proceedings, Vol 30. 1967.
Web: http://www.cs.umass.edu/~emery/classes/cmpsci691st/readings/Conc/Amdahl-04785615.pdf
[05] Anjomshoaa, Ali; Brisard, Fred y Otros: "**Job submission Description Languaje (JSDL) Specification, version 1.0**". Job submission description Language (JSDL) Specification. Global Grid Forum 2005
Web: http://www.gridforum.org/documents/GFD.56.pdf
[06] "**ANT Home Page**". Web: http://ant.apache.org/
[07] **"ARC Home Page".** Web: http://www.nordugrid.org/middleware/
[08] **"Boing Home Page".** Web: http://boinc.berkeley.edu/
[09] Buigues, Ana: "**Data Mart y Data Warehouse**".Blog . Web: http://anabuigues.com/2010/04/19/data-mart-y-data-warehouse/
[10] "**Business Process Management** : **Wikipedia**".
Web: http://en.wikipedia.org/wiki/Business_process_management
[11] Chandak, Ashish V.; Sahoo, Bibhudatta; Kumar Turuk, Ashok: "**Heuristic Task Allocation Strategies for Computational Grid**". Int. J. Advanced Networking and Applications. Vol.:02 Issue: 05, Pages: 804-810. 2011
[12] Colling, David; Ferrari, T. y Otros: "**On Quality of service support for grid computing**".2nd International Workshop on Distributed Cooperative Laboratories and Instrumenting the GRID (INGRID 2007). 2007. Web: http://pubs.doc.ic.ac.uk/ingrid-2007/ingrid-2007.pdf
[13] **"Condor Home Page".** Web: http://research.cs.wisc.edu/condor/
[14] "**Datawarehouse**". Sinnexus. Web: http://www.sinnexus.com/business_intelligence/datawarehouse.aspx
[15] Dean, Jefrey; Ghemawat, Sanjay: "**MapReduce: Simplified Data Processing on Large Clusters**". Google Inc. OSDI'04 (Symposium on Operating System Design and Implementation). San Francisco. California. 2004.
Web: http://static.googleusercontent.com/external_content/untrusted_dlcp/labs.google.com/es//papers/mapreduce-osdi04.pdf
[16] "**Definición de Procedimiento Administrativo**".
Web: http://es.wikipedia.org/wiki/Procedimiento_administrativo
[17] Del Río, Iñaki: "**Arquitectura para la Gestión de Expedientes (AIDA-GEXP)**". Ibermática S.A. 2004.
Web: http://administracionelectronica.gob.es/recursos/pae_020000321.pdf
[18] Díaz, Gilberto; Hamar, Vanessa y Otros: **"Herramientas GRID para la integración y administración de servicios de redes en Latino América"**. Corporación Parque Tecnología de Mérida. Centro de Teleinformación. Universidad de Los Andes. Web: http://www.programafrida.net/docs/informes/herramientas_grid.pdf
[19] "**EGEE Home Page**". Web: http://www.eu-egee.org/
[20]"**Expedientes Electrónicos**". AGESIC (Agencia para el Desarrollo del Gobierno de Gestión Electrónica y la Sociedad de la Información y el Conocimiento). Presidencia de Uruguay.
Web: http://www.agesic.gub.uy/innovaportal/file/540/1/fiee_1.2.pdf
[21] "**Esquemas XML para intercambio de documentos electrónicos y expedientes electrónicos: Manual de Usuario"**. Dirección General para el Impulso de la Administración Electrónica. Ministerio de Política Territorial y Administración Pública. Gobierno de España. 1ª Edición 2011.

Web:http://www.mpt.gob.es/dms/es/publicaciones/centro_de_publicaciones_de_la_sgt/Monografias0/parrafo/Manual_XML/text_es_files/Manual_esquemas-XML-intercambio-doc-exp-elec-INTERNET.pdf
[22] "**Example of the Globus alliance's Impact**" Web: http://www.globus.org/alliance/impact/
[23] Ferreira, Luis; Berstis,Viktors y Otros. "**Introduction to Grid Computing with Globus**". IBM. Sept 2003.
Web: http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf
[24] "**Formato de intercambio de expedientes electrónicos. FIEE**". Agencia para el Desarrollo del Gobierno de Gestión Electrónica la Sociedad de la Información y el Conocimiento (AGESIC). 2011.
Web:http://www.agesic.gub.uy/innovaportal/file/540/1/fiee_1.2.pdf
[25] Foster, Ian; Kesselman, Carl: "**The GRID 2**". Ed. Elsevier Inc. 2004.
Web: http://books.google.es/books?id=8-0BofIhoU0C&pg=PA355&hl=es&source=gbs_selected_pages&cad=3#v=onepage&q&f=false
[26] Fu, Xiufen; Peng, Ding; Xu, aishui, Lu, Yansheng y Xiao,Shuteng: "**Research of E-Government Information Portal applications based on Grid Technology**". The 9th International conference on Computer Supported Cooperative Work in Design Proceedings. Mayo 2005.
[27] Fuente, A.; Vázquez, J.L.; Huedo, E.; Montero, R.S.; Llorente, I.M.: **"Beneficios del uso de la tecnología grid computing en bioinformática usando la infraestructura de IRISGrid".** Boletín de RedIRIS nº 72, abril 2005.
Web: http://www.bvsde.paho.org/bvsacd/cd46/uso.pdf
[28] "**GigaSpaces Home Page**". Web: http://www.gigaspaces.com/
[29] "**gLite Home Page**". Web: http://glite.cern.ch
[30] "**Globus home page**". Web: http://www.globus.org/
[31] Gomez Velasco, Unai: "**DJ HACK Distributed Java Hash Cracker**". Memoria del proyecto fin Carrera. Departamento de Lenguajes y Sistemas Informáticos. Escuela Universitaria de Ingeniería de Vitoria-Gastaiz. Octubre 2010.
[32] Gorton, I.; Green_eld, P.; Szalay, A.; Williams, R. "**Data-Intensive Computing in the 21st Century**". IEEE Computer Society, vol. 41-4, Abril 2008, pp. 30-32.
[33] "**GridFTP**". Web: http://dev.globus.org/wiki/GridFTP
[34] **"Grid Resource allocation and Management (GRAM)"**.
Web: http://www-unix.globus.org/api/c-globus-3.2/globus_gram_documentation/html/
[35] "**HDFS Architecture Guide**". Web: http://hadoop.apache.org/common/docs/current/hdfs_design.html
[36] Herrera, Jose: "**Mobile Agents: An alternative to information exchange systems in public government**". UOC 2010. Web: http://openaccess.uoc.edu/webapps/o2/handle/10609/9021
[37] Iosup, Alexandru; Dumitrescu, Catalin; Epema, Dick y otros: "**How are Real Grids Used? The Analisys of tour Grid Traces and its implications**". Grid Computing Conference, IEEE. 2006.
[38] Iosup, Alexandru; Epema, Dick; Franke, Carsten y otros: "**On Grid Performance Evaluation using Synthetic Workloads**". CoreGRID Technical Report. Sept-2006.
[39] "**Jetty Home Page**". Web: http://jetty.codehaus.org/jetty/
[40] "**JMX Home Page**". Oracle.
Web: http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html
[41] "**JPPF. Downloads**". Web: http://www.jppf.org/downloads.php
[42] "**JPPF Home Page**". Web: http://www.jppf.org
[43] "**J2EE Connector Architecture Home Page**". Oracle. Web: http://java.sun.com/j2ee/connector/
[44] Krauter, Klaus; Buyya, Rajkumar; Maheswaran, Muthcumaru: "**A Taxonomy and Survery of Grid Resource Management Systems**". Technical Report: University of Manitoba (TR-2000/18) and Monash University (TR-2000/80). Web: http://www.buyya.com/papers/gridtaxonomy.pdf
[45] Laure,E.; Fisher, S.M. y Otros: "**Programming the Grid with gLite**". Computational Methods in Science and Technology 2006. Web: http://www.man.poznan.pl/cmst/2005/v_12_1/03Laure.pdf
[46] "**LCG Middleware Home Page**".Web: http://lcg.web.cern.ch/lcg/activities/middleware.html
[47] "**Ley 30/1992, de 26 de noviembre, de Régimen Jurídico de las Administraciones Públicas y del Procedimiento Administrativo Común**". BOE 285 de 27 noviembre 1992.
Web: http://www.boe.es/boe/dias/1992/11/27/pdfs/A40300-40319.pdf
[48] Mamoud, Qusay H.: "**Getting started with JavaSpaces technology: Beyong convencional distributed programming paradigms**" Oracle Corporation. 2005.
Web: http://java.sun.com/developer/technicalArticles/tools/JavaSpaces/
[49] "**Mantenimiento de expedientes en las Naciones Unidas**". ARMS Sección de gestión de archivos y expedientes de las Naciones Unidas.
Web: http://www.un.org/spanish/archives/unrecordsmgmt/recordkeepingintheun.shtml
[50] Mills, David: "**Internet Time Synchronization: The Network Time Protocol**". IEEE transactions on communications, Vol. 39. Web: http://networks.cs.ucdavis.edu/~mukherje/289i_sq10/mills-ntp-tcom-oct91.pdf
[51] Mills, David: "**Simple Network Time Protocol (SNTP) Version 4 form IPv4, IPv6 and OSI**". RFC2030. Web: http://gamay.tools.ietf.org/html/rfc2030
[52] Montero, R.S.; Huedo, E.; Llorente I.M.: "**Benchmarking of high throughput computing applications on Grids**". Parallel Computing. Elsevier. 2006.

[53] "**Netbeans Home Page**". Web: http://netbeans.org/
[54] "**Organización y Sistemas**". UNIDAD V- Normas y procedimientos. 2006.
Web: http://es.scribd.com/doc/54171970/2/CARACTERISTICAS-DE-LOS-FLUJOGRAMAS
[55] Peláez, José Antonio; Martín, Alfonso: "**Cuadro de mando para el análisis de la información estratégica del Ministerio de Administraciones Públicas**". BG&S Online Consultores.
Web: http://administracionelectronica.gob.es/recursos/pae_020000109.pdf
[56] Fowler, Martin: "**POJO**". Web. http://martinfowler.com/bliki/POJO.html
[57] "**Scientific Linux Home**". Web: http://www.scientificlinux.org/
[58] Shin-Iti Komats, Edson: "**Utilizando grandes computacionais no atendimento de requisitos de e-go**".
Web: http://tede.pucrs.br/tde_arquivos/4/TDE-2009-10-07T173649Z-2153/Publico/417384.pdf
[59] "**Spring Home Page**". Web: http://www.springsource.org/
[60] "**SFTP Man Page**". Web: http://www.openbsd.org/cgi-bin/man.cgi?query=sftp&sektion=1
[61] Rings, Thomas; Grabowsk, Jens; Schul, Stephan: "**A Testing Framework for Assessing Grid and Cloud Infrastructure Interoperability**". International Journal on Advances in Systems and Measurements, Vol. 4 nº 1 & 2. 2011.
Web: http://www.swe.informatik.uni-goettingen.de/sites/default/files/publications/sysmea_v4_n12_2011_8.pdf
[62] **"UNICORE Home Page".** Web: http://www.unicore.eu/
[63] "**Virtual Data Toolkit Home Page**". Web: http://vdt.cs.wisc.edu/
[64] "**Web Services Description Language (WSDL) 1.1**". Web: http://www.w3.org/TR/wsdl
[65] "**What is Mule ESB?**". Web: http://www.mulesoft.org/what-mule-esb
[66] Xiong, Jing; Wang, Jianliang; Xu, Juanliang: "**Research of Distributed Parallel Information Retrieval Based on JPPF"**. 2010 International Conference of Information Science and Management Engineering. 2010.
[67] "**Xstream Home Page**". Web: http://xstream.codehaus.org/
[68] "**Z/OS**". Web: http://es.wikipedia.org/wiki/Z/OS