Universidad del País Vasco
Euskal Herriko Unibertsitatea

Departamento de Lenguajes y Sistemas Informáticos
Escuela Universitaria de Ingeniería de Vitoria-Gasteiz

DJ HACK

DISTRIBUTED JAVA HASH CRACKER

Memoria del proyecto

presentada para optar al título de Ingeniero Técnico en Informática de Gestión

por

Unai Gómez Velasco

Director

Pablo González Nalda

10 de octubre de 2010

A mis padres
que por ellos estoy aqui

Índice general

Índice de figuras

Índice de tablas

Resumen y organización de la memoria

Resumen

Este trabajo diseña y construye una aplicación 100 % Java denominada DJ HACK (*Distributed Java HAsh craCKer*). Utilizando los planteamientos del cálculo de bases, los fundamentos de la distribución paralela y los sistemas de gestión de usuario y red, esta aplicación aporta al usuario la funcionalidad de romper claves hash de una manera totalmente segura y automatizada a través del ataque de fuerza bruta distribuido.

La aplicación va dirigida principalmente a usuarios que necesiten comprobar la seguridad de sus contraseñas mediante la rotura de claves hash. Para ello, DJ HACK desarrolla dos versiones: una server para entornos en modo carácter y otra para ordenadores con interfaz gráfica; incluyéndose, en ambas, herramientas de fácil uso, como la gestión de las claves a romper, el montaje de la red a utilizar en el ataque, y los sistemas de monitorización de tráfico.

Por otro lado, esta aplicación hace verdadero hincapié en la seguridad de su sistema. Utilizando las técnicas avanzadas de encriptación de las que dispone, adecua un cifrado robusto en los datos personales del usuario, así como, en la configuración de la base de datos. Ade-

más, incorpora soporte para poder cifrar toda la información transmitida entre los diferentes componentes del clúster.

Para todo esto, el usuario únicamente deberá registrarse en el sistema y podrá interactuar de una manera fácil y cómoda por las herramientas y funciones que posee esta innovadora aplicación.

Organización

La primera parte de esta memoria la forma el capítulo en el que se presenta el alcance del proyecto, donde se hace una descripción del mismo, se enuncian los objetivos que debe cumplir, se representa detalladamente el estudio de viabilidad realizado, riegos, etc.

La segunda parte hace un recorrido sobre la computación distribuida, los sistemas criptográficos, protocolos de comunicación que se han utilizados, etc. Además, se plantea el desarrollo técnico de la aplicación, enunciando el estudio de la tecnología realizado, y las herramientas utilizadas para poder desarrollarlo.

La tercera parte describe la aplicación DJ HACK, desarrollada en este trabajo y que expone el diseño y la estructura utilizada. Asimismo, se evalúa la gestión del proyecto mediante los retrasos que ha podido sufrir, el coste real que ha supuesto, etc. En la cuarta parte, se explica el modo de uso de la aplicación mediante su manual de usuario.

Por último, en la quinta parte se elaboran las conclusiones y se plantea el trabajo futuro.

Como información adicional, se incluyen en los anexos, el pliego de condiciones, archivos de configuración y la licencia del documento.

Parte I

Alcance del proyecto

CAPÍTULO

1 Descripción y objetivos del proyecto

Este capítulo presenta una breve descripción del proyecto resaltando sus funcionalidades básicas, así como los objetivos principales de la aplicación, de donde surgió la idea, desarrollos similares y un análisis de riegos que específica los posibles problemas que pueden surgir en la realización de la aplicación.

1.1. Descripción del proyecto

DJ HACK, acrónimo de *Distributed Java HAsh craCKer*, es una aplicación multiplataforma, multiprocesador y multitarea especialmente diseñada para intégrala y usarla en sistemas multicomputador. Dispone como objetivo fundamental la decodificación de claves hash mediante técnicas de hackeo en modo local y distribuido.

Por ello el usuario sólo debe diseñar y configurar, a través de las herramientas que proporciona la aplicación, la arquitectura a usar en el cómputo distribuido. Esta topología de red está compuesta por el cliente, los servidores, los nodos y la base de datos.

Así pues, la aplicación, entre otras funcionalidades ofrece la posibilidad de realizar dos tipos de ataque:

- Ataque de fuerza bruta[1]:

  A través de las herramientas proporcionadas por DJ HACK, el usuario puede registrar y gestionar las claves a procesar: dotándolas un título y una descripcion, determinado las longitudes posibles que puede llegar a tener la clave real, añadiendo la codificación a usar para la desencriptación de la contraseña y otorgando el alfabeto a utilizar para romperla.

- Diccionario:

  Para la utilización de esta funcionalidad no es necesario disponer de la arquitectura de distribución paralela, ya que ataque por diccionario se ejecuta en modo local debido a que no requiere una carga computacional excesiva. Para su uso el usuario únicamente debe cargar el diccionario para decodificar la clave.

Otra herramienta de la aplicación, es la posibilidad de crear los elementos del clúster de forma totalmente automatizada. El usuario puede generar y configurar cada uno de los componentes de la red especificando los puertos y conexiones a utilizar, la memoria máxima disponible, el número de núcleos a utilizar por el procesador en el ataque y su prioridad en el sistema, el cifrado de la transmisión, etc.

Además, el usuario tiene la capacidad de generar sus propias claves hash utilizando el algoritmo de encriptación que él desee.

Por otro lado, DJ HACK añade herramientas para la configuración de la base de datos (nombre de usuario, contraseña, ubicación) e incorpora, únicamente si la base de datos lo soporta, conexiones cifradas a través de SSL.

Es de esperar que tratándose de una aplicación distribuida tenga una herramienta de monitorización, gestión y seguimiento, por cada uno de los componentes del clúster. Para ello se ha integrado al programa la consola de administración y monitorización de JPPF; una aplicación encargada mostrar los niveles del tráfico en la red, la carga de las taras enviadas a cada uno de los servidores, etc.

En conclusión, DJ HACK es una aplicación 100 % *Java* totalmente innovadora, cuya principal herramienta se centra en los ataques de fuerza bruta a claves hash mediante sistemas distribuidos, proporcionando utilidades como la creación automatizada de componentes y la monitorización en tiempo real del clúster.

[1] Es necesario disponer de un entorno distribuido.

1.2. Objetivos del proyecto

El principal objetivo de este proyecto es desarrollar una aplicación capaz de realizar ataques de fuerza bruta contra contraseñas hash de manera totalmente distribuida. Para ello se deben de realizar algoritmos paralelizables y muy óptimos, capaces de reducir al máximo el coste operacional y logrando así una mayor eficiencia y rapidez en el ataque.

Por ello, el eje principal del proyecto se centra en tres pilares fundamentales: en el diseño elemental del cómputo distribuido y su organización, la arquitectura de la red a utilizar en el ataque, y los sistemas de seguridad necesarios que se enfocan en la protección de la infraestructura computacional y todo lo relacionado con esta (incluyendo la información). Lo que supone realizar una investigación de las tecnologías más eficientes y versátiles a utilizar, entender su funcionamiento y encontrar la manera de modificarlas para poder adaptarlas a las necesidades de DJ HACK.

No obstante, además serán necesarios e imprescindibles desarrollar sistemas matemáticos encargados de generar cadenas numéricas secuenciales a la hora de realizar el hackeo; sistemas de copias de seguridad (Backups) eficientes capaces de anticiparse a cualquier fallo que pueda corromper la integridad de la aplicación; y una interfaz amigable pero completa, que no requiera apenas conocimientos informáticos por parte del usuario para realizar las auditorias, así como, la implantación del sistema distribuido en el clúster y su posterior mantenimiento.

1.3. Como surgió la idea

La idea para realizar este proyecto fue sugerida por el director Pablo Gonzalez Nalda al alumno, Unai Gómez Velasco.

Se propuso explotar un framework llamado JPPF, encargado de distribuir en un clúster operaciones de gran carga computacional. Por ello se estudiaron diversos temas que requiriesen un alto rendimiento hardware, llegando a la fuerza bruta como objetivo principal. La idea de realizar una aplicación orientada a la fuerza bruta, fue elegida debido a la innovación de crear por primera vez una aplicación capaz de realizar cálculos matemáticos de un alto coste operacional de manera conjunta entre los diferentes componentes del clúster para lograr un objetivo común; el hackeo de contraseñas hash.

Por otro lado, el lenguaje de programación que más dominaba el desarrollador era *Java*, pero debido la inexperiencia total del cómputo distribuido por parte del programador y a la innovación del proyecto, se requirieron el estudio de nuevos entornos de trabajo y técnicas eficientes que se aplicaron posteriormente en el desarrollo de la aplicación.

Antes de empezar a desarrollar, se fueron definiendo las líneas base que debía seguir el proyecto respecto a su funcionalidad; como la gestión del usuario, el servicio y mantenimiento

de los distintos componentes que formarían el entorno distribuido (cliente, servidores, nodos), operaciones que debía realizar, estructura de la base de datos, sistemas de codificaciones, etc.

Aunque la primera parte del proyecto fue generar un sistema de cómputo avanzado para la distribución del trabajo entre los diferentes ordenadores, se complementó con herramientas para la monitorización y gestión del tráfico de la red, la realización ataques por diccionario, el sistema de seguridad de la aplicación, los sistemas de almacenamiento de claves y sus correspondientes copias de seguridad, etc.

Ha día de hoy, la combinación del sistema de cómputo junto con las herramientas de seguridad y gestión, han logrado realizar este proyecto: un entorno capaz de computar ataques a claves hash a través del cálculo distribuido y de la autogestión.

La aplicación ha sido desarrollada para ser instalada en cualquier sistema operativo que soporte *Java*, e integrarse principalmente en sistemas multicomputador, multiprocesador y multihilo. No obstante, también es soportada en equipos con monoprocesadores.

1.4. Desarrollos similares

Debido a la innovación del proyecto y a las tecnologías utilizas, no es posible encontrar otras aplicaciones que recojan las capacidades y funciones de DJ HACK para el hackeo distribuido de claves hash. No obstante, se han encontrado aplicaciones que tratan sobre este mismo objetivo:

- Hashcrack: Hashcrack.com es una página web que ofrece la posibilidad de romper contraseñas hash, mediante el uso de una base de datos. En dicho sistema se encuentran una elevada cantidad de valores hash pre-calculados. En comparación con DJ HACK esta página no realiza ningún ataque de fuerza bruta, ni utiliza ningún mecanismo de cómputo para encontrar la contraseña. Su metodología se basa en buscar en la base de datos la palabra hash y devolver su valor, lo que supone no poder romper la clave si esta no se encuentra entre sus sistemas.

  Dirección: `http://hashcrack.com/`

- Passcracking: Passcracking.com es una página web con la finalidad de romper contraseñas hash, utilizando una combinación de técnicas (rainbowtables clásicas, rainbowtables híbridas, diccionario) para palabras hash con encriptación MD5, y un simple diccionario para otro tipo de encriptaciones. Al igual que Hashcrack.com utiliza una base de datos donde almacena todo los valores pre-calculados. Con lo que no utiliza ningún mecanismo de fuerza bruta distribuida, menguando la posibilidad de poder encontrar la clave real de la contraseña hash enviada, ya que únicamente el usuario podrá hallar claves que se encuentren en sus sistemas.

  Dirección: `http://passcracking.com/`

CAPÍTULO

2 Viabilidad

2.1. Análisis de riegos

El análisis de riesgos es el proceso que permite la identificación de las amenazas que pueden perjudicar al desarrollo del proyecto y determinar el impacto o grado de perjuicio que pueden ocasionar. Implica analizar las amenazas y vulnerabilidades que puedan darse y se valoran en términos de probabilidad de ocurrencia y gravedad de impacto sobre el proyecto.

A continuación se establece una escala (de menor a mayor gravedad) para valorar el impacto que causarían los riesgos que se describen a continuación:

- *Muy grave*: De consecuencias fatales para el desarrollo del proyecto, requeriría replantearse, incluso, si merece la pena continuar con el proyecto.
- *Grave*: Problemas que podrían imposibilitar terminar el proyecto en el plazo previsto.

Requeriría realizar una profunda re planificación ampliando el número de horas a dedicar o reduciendo el trabajo a realizar.

- *Perjudicial*: Pequeños retrasos relativamente sencillos de solventar, aunque pueden resultar peligrosos si no se identifican y subsanan a tiempo.
- *Leve*: Incidencia de poca importancia que en ningún caso compromete el desarrollo normal del proyecto.

Los posibles riesgos que se tienen en cuenta en la realización del proyecto los siguientes:

Elección equivocada de las herramientas de trabajo:

- *Descripción*: Es posible que con alguna de las tecnologías elegidas no sea posible cumplir los objetivos planteados o dar a la aplicación la funcionalidad deseada.
- *Probabilidad*: 8 %
- *Alcance*: Muy grave, ya que supondría replantear por completo el proyecto.
- *Medidas preventivas:* Realizar un estudio profundo sobre las diferentes tecnologías y tener planteadas tecnologías alternativas.
- Medidas correctoras: Comenzar la implementación con otras tecnologías en el menor tiempo posible.

Desconocimiento del software elegido:

- *Descripción*: En este proyecto se van a usar varias herramientas con las que el desarrollador no está nada familiarizado y esto podría ocasionar retrasos a la hora de realizar la aplicación.
- *Probabilidad*: 100 %
- *Alcance*: Perjudicial, impediría avanzar en el tiempo estimado.
- *Medidas preventivas*: Planificar una correcta formación en el uso de las tecnologías que sean nuevas y tener preparado de antemano documentación al respecto.
- *Medidas correctoras*: Estudiar la documentación y usar ejemplos para adaptarse a las nuevas herramientas en un tiempo breve.

Estancamiento en la codificación:

- *Descripción*: Podría llegar a darse algún problema de difícil solución que paralizaría momentáneamente el avance del proyecto.
- *Probabilidad*: 47 %
- *Alcance*: Perjudicial, impediría avanzar en el tiempo estimado.

- *Medidas preventivas*: Tener bien estudiadas las herramientas de trabajo y ajustar los plazos a las necesidades del desarrollador.
- *Medidas correctoras*: Replanificar las tareas posteriores y dedicar más formación a las herramientas.

Pérdida de los datos del proyecto por fallo del hardware o software maligno:

- *Descripción*: Debido a un virus o a un fallo en el disco duro del ordenador en el que se desarrolla el proyecto se podrían perder todos los datos.
- *Probabilidad*: 13 %
- *Alcance*: Muy grave, pues supondría empezar de cero.
- *Medidas preventivas*: Tener copias de seguridad del proyecto en distintos ordenadores y dispositivos externos e instalar un antivirus y cortafuegos en el sistema.
- *Medidas correctoras*: Intentar recuperar la información de los medios de almacenamiento.

Problema con los recursos materiales:

- *Descripción*: Los ordenadores usados para desarrollar el proyecto pueden no superar el rendimiento necesario para realizar la aplicación e incluso sufrir algún percance que los deje inservibles temporal o permanentemente.
- *Probabilidad*: 10 %
- *Alcance*: Perjudicial, ya que retrasarían el proyecto de una forma indefinida aunque no impedirían su desarrollo.
- *Medidas preventivas*: Realizar un mantenimiento frecuente de las computadoras así como hacer uso correcto de ellas y tener preparados más dispositivos que se puedan usar como alternativa en caso de emergencia.
- *Medidas correctoras*: Buscar una solución rápidamente ya sea reparando las computadoras o adquiriendo otras nuevas.

Baja del desarrollador de la aplicación:

- *Descripción*: Podría suceder que por enfermedad o accidente, el desarrollador del proyecto quede incapacitado durante una temporada.
- *Probabilidad*: 5 %
- *Alcance*: Perjudicial, puesto que no impide la realización del proyecto pero sí lo puede retrasar.
- *Medidas preventivas*: No previsible.

- *Medidas correctoras*: Dependiendo de la gravedad del problema sería posible continuar avanzando en el proyecto.

Análisis o planificación muy optimistas:

- *Descripción*: Una mala planificación podría acarrear retrasos en el desarrollo del proyecto.
- *Probabilidad*: 17 %
- *Alcance*: Perjudicial, supondría un retraso del proyecto respecto a la planificación.
- *Medidas preventivas*: Realizar una planificación los más realista posible sin estar demasiado ajustada y teniendo en cuenta las probabilidades de los riegos que se están analizando.
- *Medidas correctoras*: Modificar la planificación alargando la duración del proyecto o reduciendo las pretensiones iniciales.

2.2. Planificación del tiempo del proyecto

Este apartado describe la estructura de descomposición del trabajo utilizada en la realización de la aplicación; las fases, tareas y entregables del proyecto; así como, la agenda de trabajo y los recursos materiales empleados.

2.2.1. Estructura de descomposición del trabajo

La EDT (Estructura de descomposición del trabajo) pretende estructurar de manera exhaustiva, jerárquica y descendente los entregables y las tareas necesarias para completar el proyecto. Por tanto, la planificación del proyecto quedaría según las figuras 2.1 y 2.2.

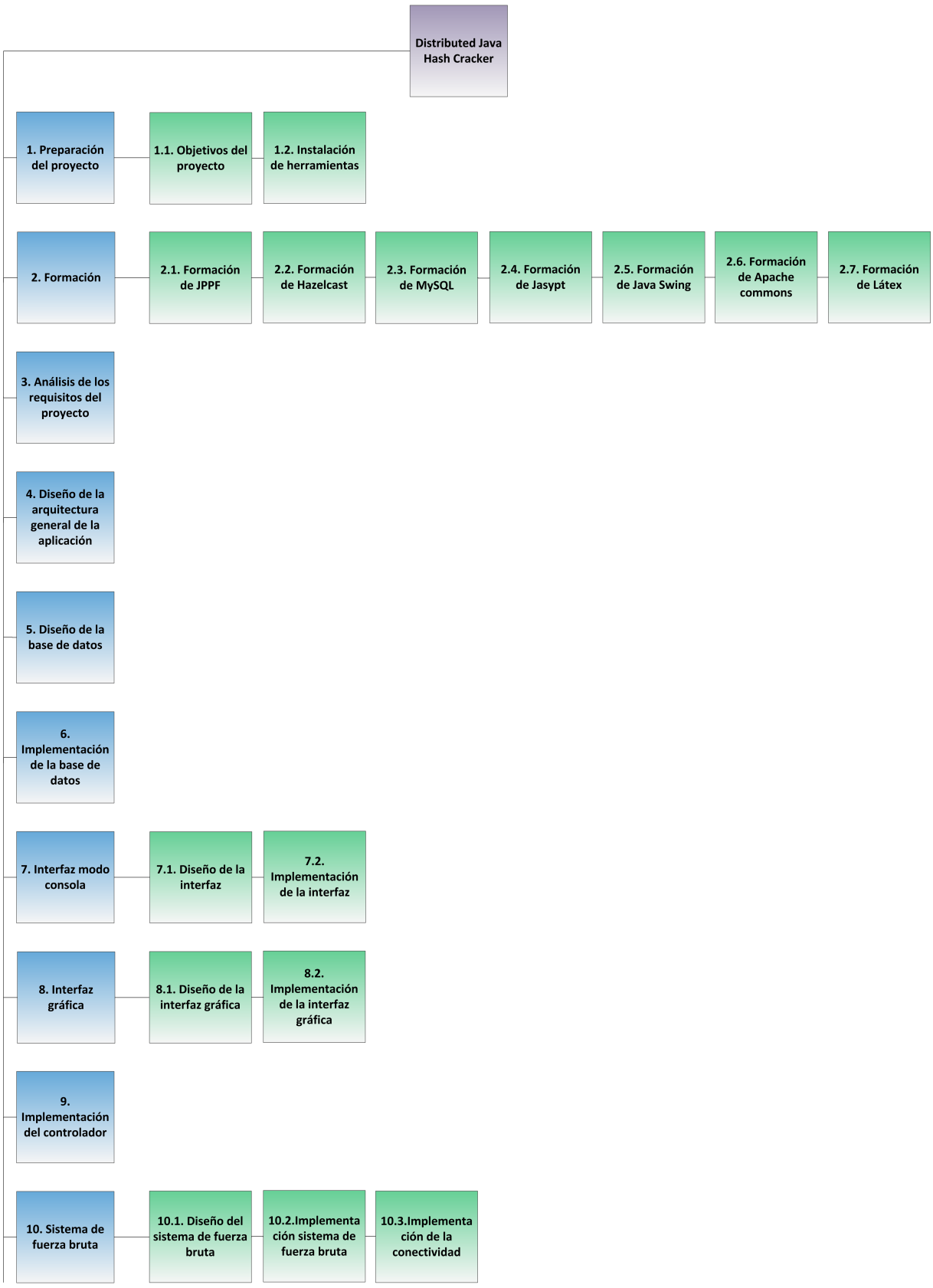


Figura 2.1: Estructura de descomposición del trabajo I.

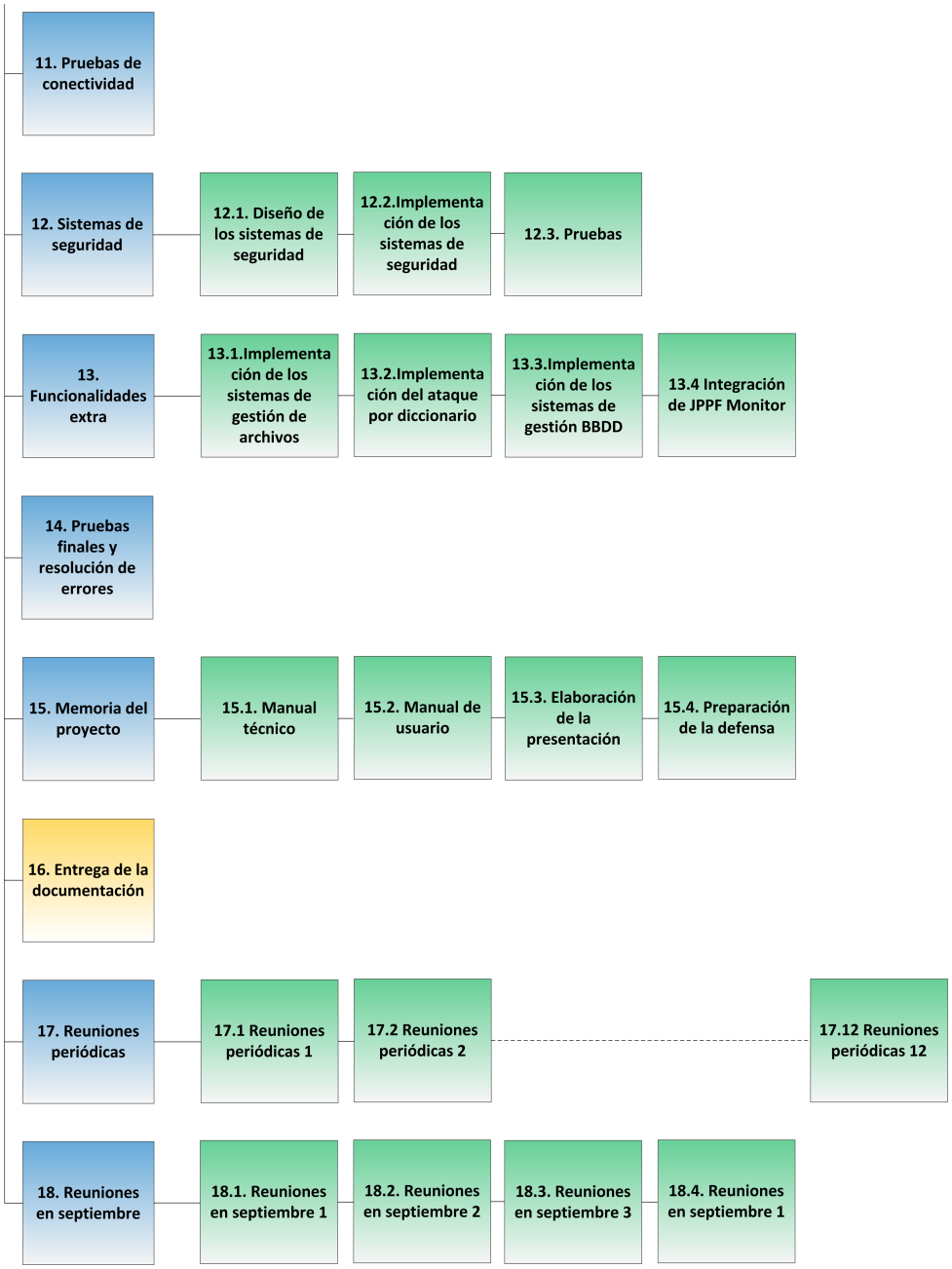


Figura 2.2: Estructura de descomposición del trabajo II.

2.2.2. Fases, tareas y entregables

Esta sección ilustra y describe las fases, tares y entregables planificados en la realización del proyecto.

2.2.2.1. Listado de fases tareas y entregables

A continuación se muestra el listado de fases y tareas de las que dispondrá el proyecto:

0 Distributed Java Hash Cracker

1 Preparación del proyecto

1.1 Objetivos del proyecto

1.2 Instalación de herramientas

2 Formación

2.1 Formación de JPPF

2.2 Formación de Hazelcast

2.3 Formación MySQL

2.4 Formación de Jasypt

2.5 Formación de Java Swing

2.6 Formación de Apache commons

2.7 Formación de Látex

3 Análisis de los requisitos del proyecto

4 Diseño de la arquitectura general de la aplicación

5 Diseño de la base de datos

6 Implementación de la base de datos

7 Interfaz modo consola

7.1 Diseño de la interfaz

7.2 Implementación de la interfaz

8 Interfaz gráfica

8.1 Diseño de la interfaz gráfica

8.2 Implementación de la interfaz gráfica

9 Implementación del controlador

10 Sistema de fuerza bruta

10.1 Diseño del sistema de fuerza bruta

10.2 Implementación del sistema de fuerza bruta

10.3 Implementación de la conectividad

11 Pruebas de conectividad

12 Sistemas de seguridad

12.1 Diseño de los sistemas de seguridad

12.2 Implementación de los sistemas de seguridad
12.3 Pruebas
13 Funcionalidades extra
13.1 Implementación de los sistemas de gestion de archivos
13.2 Implementación del ataque por diccionario
13.3 Implementación de los sistemas de gestion de BBDD
13.4 Integración de JPPF Monitor
14 Pruebas finales y solución de errores
15 Memoria del proyecto
15.1 Manual técnico
15.2 Manual de usuario
15.3 Elaboración de la presentación
15.4 Prepararción de la defensa
16 Entrega de la documentación
17 Reuniones periódicas
17.1 Reuniones periódicas 1
17.2 Reuniones periódicas 2
17.3 Reuniones periódicas 3
17.4 Reuniones periódicas 4
17.5 Reuniones periódicas 5
17.6 Reuniones periódicas 6
17.7 Reuniones periódicas 7
17.8 Reuniones periódicas 8
17.9 Reuniones periódicas 9
17.10 Reuniones periódicas 10
17.11 Reuniones periódicas 11
17.12 Reuniones periódicas 12
18 Reuniones en septiembre
18.1 Reuniones en septiembre 1
18.2 Reuniones en septiembre 2
18.3 Reuniones en septiembre 3
18.4 Reuniones en septiembre 4

2.2.2.2. Descripcion detallada de las tareas y entregables

A continuación se muestra una ficha para cada tarea identificada, donde se anota su número, nombre, una breve descripción y el trabajo estimado:

Nombre: Objetivos del proyecto.

Número: 1.1

Descripción: Hacer un análisis previo para determinar los objetivos que se intentarán alcanzar con este proyecto.

Trabajo estimado: 8 horas.

Número: 1.2

Nombre: Instalación de herramientas.

Descripción: Instalar en cada uno de los ordenadores las herramientas necesarias para el desarrollo del proyecto.

Trabajo estimado: 8 horas.

Número: 2.1

Nombre: Formación de JPPF.

Descripción: Realizar un estudio exhaustivo para familiarizarse con la herramienta y adquirir los conocimientos necesarios del framework JPPF.

Trabajo estimado: 30 horas.

Número: 2.2

Nombre: Formación de Hazelcast.

Descripción: Realizar un estudio exhaustivo para familiarizarse con la herramienta y adquirir los conocimientos necesarios del framework Hazelcast.

Trabajo estimado: 9 horas.

Número: 2.3

Nombre: Formación MySQL.

Descripción: Realizar un repaso de MySQL orientado al pool de conexiones.

Trabajo estimado: 4 horas.

Número: 2.4

Nombre: Formación de Jasypt.

Descripción: Realizar un estudio detallado para familiarizarse con la herramienta y adquirir los conocimientos necesarios de la biblioteca Jasypt.

Trabajo estimado: 7 horas.

Número: 2.5

Nombre: Formación de Java Swing.

Descripción: Adquirir los conocimientos necesarios de la herramienta para poder realizar una interfaz gráfica amigable.

Trabajo estimado: 16 horas.

Número: 2.6

Nombre: Formación de Apache commons.

Descripción: Analizar las librerías commons Lang, commons DBCP, commons Pool, commons Codec, commons Logging que serán necesarias para realizar la aplicación.

Trabajo estimado: 4 horas.

Número: 2.7

Nombre: Formación de Látex.

Descripción: Familiarizarse con latex para poder redactar la memoria del proyecto.

Trabajo estimado: 11 horas.

Número: 3

Nombre: Análisis de los requisitos del proyecto.

Descripción: Definición de los requisitos funcionales que va a tener el proyecto para cumplir los objetivos.

Trabajo estimado: 10 horas.

Número: 4

Nombre: Diseño de la arquitectura general de la aplicación.

Descripción: Especificaciones de la estructura interna de la aplicación, definir la especificación MVC, etc.

Trabajo estimado: 15 horas.

Número: 5

Nombre: Diseño de la base de datos.

Descripción: Determinar cuál va a ser la estructura de la base de datos, numero de tablas, atributos de cada una y relaciones.

Trabajo estimado: 3 horas.

Número: 6

Nombre: Implementación de la base de datos.

Descripción: Implementación en SQL de la base de datos según las especificaciones que se han hecho en el diseño.

Trabajo estimado: 3 horas.

Número: 7.1

Nombre: Diseño de la interfaz.

Descripción: Se especifica cómo va a ser el entorno visual en modo carácter por el cual se van a mover los usuarios.

Trabajo estimado: 4 horas.

Número: 7.2

Nombre: Implementación de la interfaz.

Descripción: Implementación de la interfaz siguiendo el diseño realizado.

Trabajo estimado: 16 horas.

Número: 8.1

Nombre: Diseño de la interfaz gráfica.

Descripción: Se especifica cómo va a ser el entorno gráfico por el que se van a mover los usuarios.

Trabajo estimado: 4 horas.

Número: 8.2

Nombre: Implementación de la interfaz gráfica.

Descripción: Implementación de la interfaz gráfica siguiendo el diseño realizado.

Trabajo estimado: 103 horas.

Número: 9

Nombre: Implementación del controlador.

Descripción: Implementación del sistema encargado de conectar la interfaz gráfica con el núcleo del sistema.

Trabajo estimado: 9 horas.

Número: 10.1

Nombre: Diseño del sistema de fuerza bruta.

Descripción: Se especifica cómo se va a estructurar el sistema de fuerza bruta, el diseño de la librerías matemáticas necesarias a implementar, la metodología a emplear para realizar ataques distribuidos, el diseño de las conexiones, etc.

Trabajo estimado: 41 horas.

Número: 10.2

Nombre: Implementación del sistema de fuerza bruta.

Descripción: Implementación del sistema de fuerza bruta siguiendo el diseño realizado.

Trabajo estimado: 162 horas.

Número: 10.3

Nombre: Implementación de la conectividad.

Descripción: Implementación de la conectividad entre los diferentes miembros del clúster para realizar un ataque de fuerza bruta siguiendo el diseño realizado.

Trabajo estimado: 60 horas.

Número: 11

Nombre: Pruebas de conectividad.

Descripción: Plan de pruebas para verificar el correcto funcionamiento de la recepción y envío de datos entre los componentes del clúster.

Trabajo estimado: 12 horas.

Número: 12.1

Nombre: Diseño de los sistemas de seguridad.

Descripción: Se especifican las metodologías, algoritmos y funciones necesarias para realizar comunicaciones seguras en la red, así como la encriptación de ficheros, autenticación, etc.

Trabajo estimado: 40 horas.

Número: 12.2

Nombre: Implementación de los sistemas de seguridad.

Descripción: Implementación de los sistemas de seguridad siguiendo la especificación realizada.

Trabajo estimado: 84 horas.

Número: 12.3

Nombre: Pruebas.

Descripción: Realización pruebas necesarias para comprobar el correcto funcionaiento de los sistemas de seguridad a traves de ténicas de sniffing.

Trabajo estimado: 20 horas

Número: 13.1

Nombre: Implementación de los sistemas de gestión de archivos.

Descripción: Desarrollo de los sistemas encargados del tratamiento de archivos, creación de nodos y drivers, etc.

Trabajo estimado: 38 horas.

Número: 13.2

Nombre: Implementación del ataque por diccionario.

Descripción: Programación de un sistema capaz de realizar ataques de diccionario.

Trabajo estimado: 5 horas.

Número: 13.3

Nombre: Implementación de los sistemas de gestión de BBDD.

Descripción: Desarrollo de un sistema encargado de interactuar con la base de datos de una manera segura y tolerante a fallos.

Trabajo estimado: 7 horas.

Número: 13.4

Nombre: Integración de JPPF Monitor.

Descripción: Integración de la consola proporcionado por JPPF para supervisar el tráfico de la red, gestionar componentes del clúster, etc.

Trabajo estimado: 4 horas.

Número: 14

Nombre: Pruebas finales y solución de errores.

Descripción: Plan de pruebas definitivo para comprobar la correcta interacción de todos los elementos del proyecto y el buen funcionamiento de las comunicaciones.

Trabajo estimado: 16 horas.

Número: 15.1

Nombre: Manual técnico.

Descripción: Redactar un manual que especifique como instalar la aplicación.

Trabajo estimado: 70 horas.

Número: 15.2

Nombre: Manual de usuario.

Descripción: Redactar una manual que detalle al usuario como usar la aplicación y detalle todas sus funcionalidades.

Trabajo estimado: 17 horas.

Número: 15.3

Nombre: Elaboración de la presentación.

Descripción: Elaborar las diapositivas necesarias para la defensa del proyecto.

Trabajo estimado: 16 horas.

Número: 15.4

Nombre: Preparación de la defensa.

Descripción: Realizar ensayos previos a la presentación para conseguir una buena exposición final del proyecto.

Trabajo estimado: 10 horas.

Número: 15.4

Nombre: Entrega de la documentación.

Trabajo estimado: 0 horas.

Número: 17.1 - 17.11

Nombre: Reuniones periódicas.

Descripción: Reuniones con el director cada primer martes de cada mes, para organizar el proyecto, plantear problemas y soluciones, etc.

Trabajo estimado: 3 horas por reunión.

Número: 18.1 - 18.4

Nombre: Reuniones periódicas.

Descripción: Reuniones semanales todos los lunes con el director en el mes de septiembre, para subsanar pequeños problemas, plantear sugerencias y dudas, etc.

Trabajo estimado: 2 horas por reunión.

2.2.3. Recursos humanos y materiales

Como recurso humano para la realización de este proyecto se dispone de un único desarrollador (Unai Gómez Velasco).

Como recursos materiales, se han establecido 2 ordenadores de sobremesa clónicos sin sistema operativo integrado que aportan modularidad, genericidad y escalabilidad al hardware; así como dos portátiles personales con sistema operativo integrado, por lo que la licencia del sistema operativo va incluida en el precio total de cada portátil. Además se dispone de un switch y de un router para interconectar los distintos dispositivos.

A continuación se muestra una lista detallada de las características de cada uno de los recursos:

PC Sobremesa 1:

Modelo: Clónico.

Amd Athlon(TM) 500 MHz (100x5.0).

RAM: 125MB.

HD: Fujitsu MEP3102AT 10GB.

Tarjeta gráfica: VGA integrada.

S.O. Xubuntu 9.10.

PC Sobremesa 2:

Modelo: Clónico.

Intel Pentium IV 2.53GHz (133x19).

RAM: 2GB SDRAM.

HD: Maxtor STM3250310AS IDE 250GB ID.

HD2: ST3120022A 120GB.

Tarjeta gráfica: ATI Radeon 9800pro.

S.O. Kubuntu 10.4

Netbook:

Modelo: Netway Iridum U20.

Intel Atom Processor N270 1.60 GHz - FSB 533 MHz - 512 KB Caché L2.

Tarjeta gráfica: Intel GSE 945.

RAM: 1024 MB soDIMM DDR2 533/667.

HD: SATA 250 GB.

Gráfica: VGA integrada.

S.O. Windows XP Profesional.

Portátil:

Modelo: Alienware M15x.

Intel core i7 CPU Q720 1.6GHz.

RAM: DDR3 SDRAM de 4.096 MB a 1.333 MHz canal dual.

HD: SATA de 500 GB (7.200 rpm).

Gráfica: 12MB NVIDIA GeForce GT240M.

S.O. Windows 7 Home Premium.

Switch:

Modelo: OvisLink 8-port Fast Ethernet.

Velocidad de trasmisión: 10/100 Mbps o 20/200 Mbps en modo full-duplex.

Función autosensing.

Método de trasmisión: Store and forware.

Soporta autonegociación.

Router Wifi:

Modelo Telsey CPVA.

Velocidad de trasmisión: 10/100 Mbps.

Voz gestionada por VoIP a través del router.

4 puertos RJ45.

1 puerto USB.

2.2.4. Asignación de recursos y agenda del proyecto

Se estima que la realización del proyecto durará del día 1 de septiembre de 2009 al 27 de septiembre del 2010. Esta planificación está adaptada a las necesidades del desarrollador ya que deberá compatibilizar el proyecto con las clases en la universidad. Por otra parte, se deja margen para llegado el caso de requeir más tiempo del previsto a alguna tarea, haya posiblidad de realizarla.

Siguiendo el calendario laboral 2.1 oficial BOE para Álava-Araba se han establecido los siguientes días como festivos, además 11 días de vacaciónes en la segunda quincena de agosto.

Fecha	Evento
1 de enero	Año nuevo
6 de enero	Día de Reyes
19 de marzo	San José
1 de abril	Jueves Santo
2 de abril	Viernes Santo
5 de abril	Lunes de Pascua de Resurección
28 de abril	San Prudencio
1 de mayo	Fiesta del Trabajo
5 de agosto	Virgen Blanca
15 de agosto 2010 - 30 de agosto 2010	Vaciones de verano
12 de octubre	Fiesta Nacional de España
1 de noviembre	Todos los Santos
6 de diciembre	Día de la Constitución Española
8 de diciembre	La Inmaculada
25 de diciembre	Navidad

Tabla 2.1: Calendario laboral.

A continuación se muestra la lista de las tareas con la asignación de recursos y el trabajo estimado de cada una *(Véase las tablas 2.2 y 2.3)*, así como el diagrama de Gantt 2.3 y 2.4. De dichas tablas se han obtienido los siguientes datos:

Trabajo total estimado: 888 horas

Duración total estimada: 1030 horas

Coste estimado: 4.762,80 €

2.2.4.1. Panificación

A continuación se muestra el listado de fases y tareas de las que dispondrá el proyecto, así como los recursos asignados a cada tarea.

EDT	Nombre de tarea	Duración	Trabajo	Costo	Comienzo	Fin
0	**Distributed Java Hash Cracker**	**1030 horas**	**888 horas**	**4.755,90 €**	**mar 01/09/09**	**lun 27/09/10**
1	**Preparación del proyecto**	**16 horas**	**16 horas**	**90,80 €**	**mar 01/09/09**	**vie 04/09/09**
1.1	Objetivos del proyecto	8 horas	8 horas	42,40 €	mar 01/09/09	mié 02/09/09
1.2	Instalación de herramientas	8 horas	8 horas	48,40 €	jue 03/09/09	vie 04/09/09
2	**Formación**	**81 horas**	**81 horas**	**439,80 €**	**lun 07/09/09**	**lun 05/10/09**
2.1	Formación de JPPF	30 horas	30 horas	160,50 €	lun 07/09/09	mié 16/09/09
2.2	Formación de Hazelcast	9 horas	9 horas	49,20 €	mié 16/09/09	vie 18/09/09
2.3	Formación MySQL	4 horas	4 horas	22,70 €	vie 18/09/09	lun 21/09/09
2.4	Formación de Jasypt	7 horas	7 horas	38,60 €	lun 21/09/09	mié 23/09/09
2.5	Formación de Java Swing	16 horas	16 horas	86,30 €	mié 23/09/09	mar 29/09/09
2.6	Formación de Apache commons	4 horas	4 horas	22,70 €	mar 29/09/09	mié 30/09/09
2.7	Formación de Látex	11 horas	11 horas	59,80 €	mié 30/09/09	lun 05/10/09
3	Análisis de los requisitos del proyecto	10 horas	10 horas	53,00 €	lun 05/10/09	mié 07/10/09
4	Diseño de la arquitectura general de la aplicación	15 horas	15 horas	79,50 €	mié 07/10/09	mié 14/10/09
5	Diseño de la base de datos	3 horas	3 horas	15,90 €	mié 14/10/09	jue 15/10/09
6	Implementación de la base de datos	3 horas	3 horas	17,40 €	jue 15/10/09	jue 15/10/09
7	**Interfaz modo consola**	**20 horas**	**20 horas**	**107,50 €**	**vie 16/10/09**	**jue 22/10/09**
7.1	Diseño de la interfaz	4 horas	4 horas	21,20 €	vie 16/10/09	vie 16/10/09
7.2	Implementación de la interfaz	16 horas	16 horas	86,30 €	lun 19/10/09	jue 22/10/09
8	**Interfaz gráfica**	**107 horas**	**107 horas**	**567,10 €**	**vie 23/10/09**	**lun 30/11/09**
8.1	Diseño de la interfaz gráfica	4 horas	4 horas	21,20 €	vie 23/10/09	vie 23/10/09
8.2	Implementación de la interfaz gráfica	103 horas	103 horas	545,90 €	lun 26/10/09	lun 30/11/09
9	Implementación del controlador	9 horas	9 horas	49,20 €	lun 30/11/09	mié 02/12/09
10	**Sistema de fuerza bruta**	**263 horas**	**263 horas**	**1.396,09 €**	**jue 03/12/09**	**mié 10/03/10**
10.1	Diseño del sistema de fuerza bruta	41 horas	41 horas	217,30 €	jue 03/12/09	vie 18/12/09
10.2	Implementación del sistema de fuerza bruta	162 horas	162 horas	860,10 €	vie 18/12/09	mié 17/02/10
10.3	Implementación de la conectividad	60 horas	60 horas	319,50 €	mié 17/02/10	mié 10/03/10
11	Pruebas de conectividad	12 horas	12 horas	69,60 €	mié 10/03/10	lun 15/03/10

Tabla 2.2: Listado de fases tareas y entregables I.

EDT	Nombre de tarea	Duración	Trabajo	Costo	Comienzo	Fin
12	**Sistemas de seguridad**	**144 horas**	**144 horas**	**770,70 €**	**lun 15/03/10**	**mar 11/05/10**
12.1	Diseño de los sistemas de seguridad	40 horas	40 horas	212,00 €	lun 15/03/10	mar 30/03/10
12.2	Implementación de los sistemas de seguridad	84 horas	84 horas	446,70 €	mar 30/03/10	mar 04/05/10
12.3	Pruebas	20 horas	20 horas	112,00 €	mar 04/05/10	mar 11/05/10
13	**Funcionalidades extra**	**54 horas**	**54 horas**	**292,20 €**	**mar 11/05/10**	**lun 31/05/10**
13.1	Implementación de los sistemas de gestion de archivos	38 horas	38 horas	202,90 €	mar 11/05/10	mar 25/05/10
13.2	Implementación del ataque por diccionario	5 horas	5 horas	28,00 €	mar 25/05/10	mié 26/05/10
13.3	Implementación de los sistemas de gestion de BBDD	7 horas	7 horas	38,60 €	mié 26/05/10	vie 28/05/10
13.4	Integración de JPPF Monitor	4 horas	4 horas	22,70 €	vie 28/05/10	lun 31/05/10
14	Pruebas finales y solución de errores	16 horas	16 horas	86,30 €	lun 31/05/10	vie 04/06/10
15	**Memoria del proyecto**	**96 horas**	**113 horas**	**603,40 €**	**vie 04/06/10**	**jue 08/07/10**
15.1	Manual ténico	70 horas	70 horas	372,50 €	vie 04/06/10	mar 29/06/10
15.2	Manual de usuario	17 horas	17 horas	90,10 €	mar 29/06/10	lun 05/07/10
15.3	Elaboración de la presentación	16 horas	16 horas	86,30 €	mar 29/06/10	lun 05/07/10
15.4	Prepararción de la defensa	10 horas	10 horas	54,50 €	lun 05/07/10	jue 08/07/10
16	Entrega de la documentación	0 horas	0 horas	0,00 €	jue 08/07/10	jue 08/07/10
17	**Reuniones peródicas**	**923 horas**	**18 horas**	**95,40 €**	**mar 01/09/09**	**mar 03/08/10**
17.1	Reuniones peródicas 1	3 horas	1,5 horas	7,95 €	mar 01/09/09	mar 01/09/09
17.2	Reuniones peródicas 2	3 horas	1,5 horas	7,95 €	mar 06/10/09	mar 06/10/09
17.3	Reuniones peródicas 3	3 horas	1,5 horas	7,95 €	mar 03/11/09	mar 03/11/09
17.4	Reuniones peródicas 4	3 horas	1,5 horas	7,95 €	mar 01/12/09	mar 01/12/09
17.5	Reuniones peródicas 5	3 horas	1,5 horas	7,95 €	mar 05/01/10	mar 05/01/10
17.6	Reuniones peródicas 6	3 horas	1,5 horas	7,95 €	mar 02/02/10	mar 02/02/10
17.7	Reuniones peródicas 7	3 horas	1,5 horas	7,95 €	mar 02/03/10	mar 02/03/10
17.8	Reuniones peródicas 8	3 horas	1,5 horas	7,95 €	mar 06/04/10	mar 06/04/10
17.9	Reuniones peródicas 9	3 horas	1,5 horas	7,95 €	mar 04/05/10	mar 04/05/10
17.10	Reuniones peródicas 10	3 horas	1,5 horas	7,95 €	mar 01/06/10	mar 01/06/10
17.11	Reuniones peródicas 11	3 horas	1,5 horas	7,95 €	mar 06/07/10	mar 06/07/10
17.12	Reuniones peródicas 12	3 horas	1,5 horas	7,95 €	mar 03/08/10	mar 03/08/10
18	**Reuniones en septiembre**	**62 horas**	**4 horas**	**21,20 €**	**lun 06/09/10**	**lun 27/09/10**
18.1	Reuniones en septiembre 1	2 horas	1 hora	5,30 €	lun 06/09/10	lun 06/09/10
18.2	Reuniones en septiembre 2	2 horas	1 hora	5,30 €	lun 13/09/10	lun 13/09/10
18.3	Reuniones en septiembre 3	2 horas	1 hora	5,30 €	lun 20/09/10	lun 20/09/10
18.4	Reuniones en septiembre 4	2 horas	1 hora	5,30 €	lun 27/09/10	lun 27/09/10

Tabla 2.3: Listado de fases tareas y entregables II.

Nombre de tarea	Nombres de los recursos
Distributed Java Hash Cracker	
Preparación del proyecto	
Objetivos del proyecto	Desarollador
Instalación de herramientas	PC Sobremesa 1[1];PC Sobremesa 2[1];Netbook[1];Portátil[1];Desarollador
Formación	
Formación de JPPF	Desarollador;Portátil[1]
Formación de Hazelcast	Desarollador;Portátil[1]
Formación MySQL	Desarollador;Portátil[1]
Formación de Jasypt	Desarollador;Portátil[1]
Formación de Java Swing	Desarollador;Portátil[1]
Formación de Apache commons	Desarollador;Portátil[1]
Formación de Látex	Desarollador;Portátil[1]
Análisis de los requisitos del proyecto	Desarollador
Diseño de la arquitectura general de la aplicación	Desarollador
Diseño de la base de datos	Desarollador
Implementación de la base de datos	Desarollador;Portátil[1]
Interfaz modo consola	
Diseño de la interfaz	Desarollador
Implementación de la interfaz	Desarollador;Portátil[1]
Interfaz gráfica	
Diseño de la interfaz gráfica	Desarollador
Implementación de la interfaz gráfica	Desarollador
Implementación del controlador	Desarollador;Portátil[1]
Sistema de fuerza bruta	
Diseño del sistema de fuerza bruta	Desarollador
Implementación del sistema de fuerza bruta	Desarollador;Portátil[1]
Implementación de la conectividad	Desarollador;Portátil[1]
Pruebas de conectividad	Desarollador;Netbook[1];PC Sobremesa 1[1];PC Sobremesa 2[1];Portátil[1]

Tabla 2.4: Asignación de recursos a tareas I.

Como se puede ver en las tablas 2.2 y 2.3 la planificación comienza con una tarea repetitiva que será convocada cada el primer martes de cada més hasta finales de agosto. En el mes de septiembre se realizarán todos los lunes de cada semana convocatiorias con el director para plantear objetivos, realizar un seguimiento y supervisar el desarrollo del proyecto.

2.2.4.2. Diagrama de Gantt

El diagrama de Gantt 2.3 y 2.4 muestra el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo del tiempo total determinado para el proyecto.

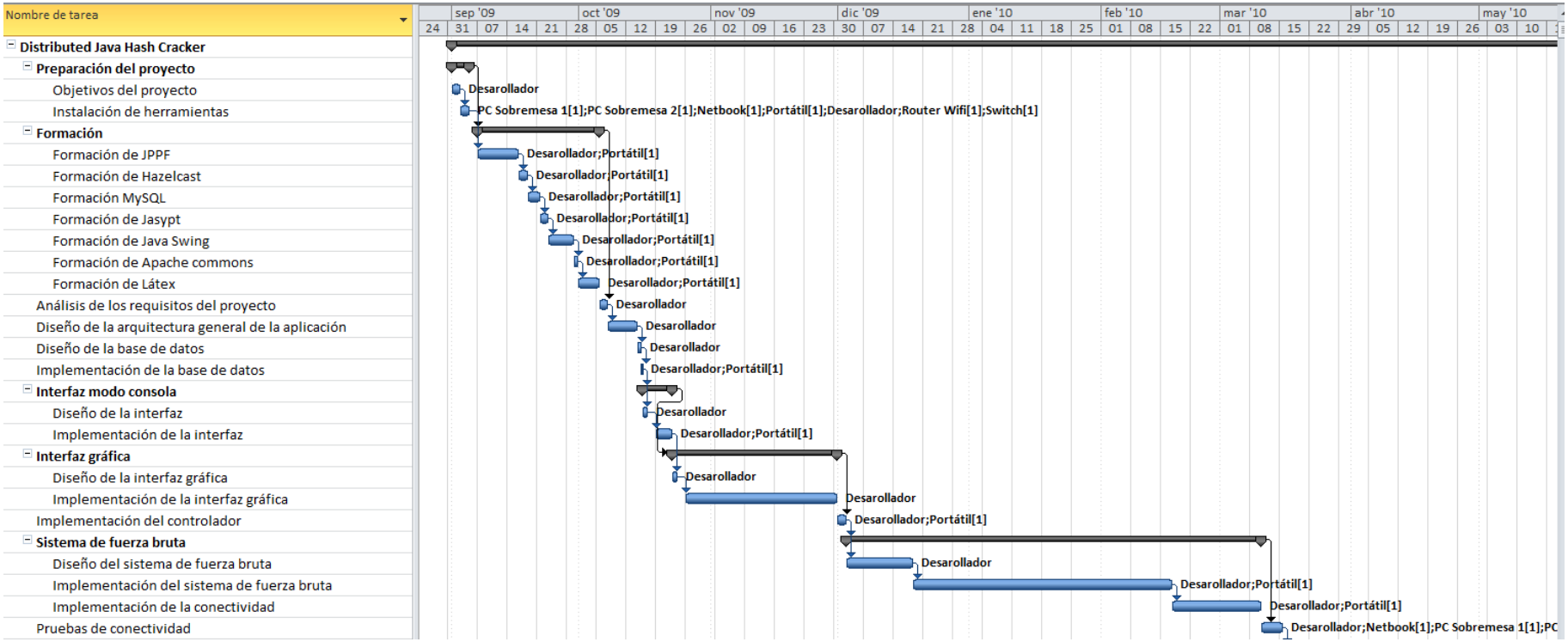


Figura 2.3: Diagrama de Gantt I.

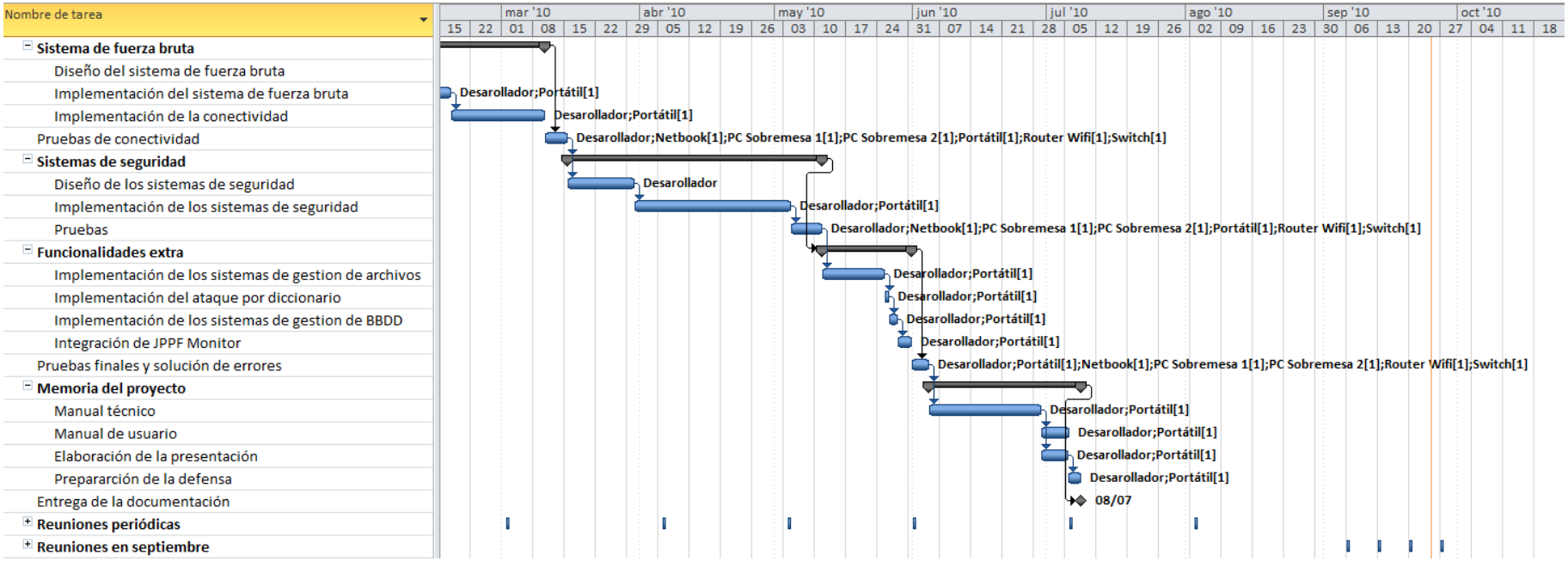


Figura 2.4: Diagrama de Gantt I.

2.3. Estimación de costes del proyecto

Para realizar la estimación de costos se toman como referencia los datos que aparecen en las tablas adjuntas. En ellas se indica lo que cobra cada uno de los distintos perfiles de los recursos humanos. Tienen una tasa estándar y una tasa de horas extra que es en todos los casos s la tasa estándar más un 10 %. La tasa estándar se ha calculado a partir de las tablas de sueldos del CEDS (Centro de Estudios y Diseño de Sistemas) con los siguientes datos:

Analista informático: 41000€÷ 14 pagas ÷ 20 días ÷ 8 horas = 18,30 €/h

Los recursos materiales se consideran cada uno de ellos con un coste por uso de 1,50€en concepto de luz y otros gastos que se cobran independientemente del tiempo de uso.

En la realización de este presupuesto se tienen en cuenta tanto los recursos materiales como las licencias de software necesarias para el desarrollo del proyecto.

2.3.1. Recursos de trabajo

La tabla 2.5 recursos de trabajo, muestra los trabajadores del proyecto así como su perfil y el importe que genera cada uno de ellos.

Concepto	**Tasa estándar**	**Tasa horas extra**
Desarollador	5,30 €/hora	7,15 €/hora

Tabla 2.5: Recursos de trabajo.

2.3.2. Recursos materiales

En las tablas 2.6 y 2.7, se muestran los recursos materiales hardware y software empleados en el proyecto.

Concepto	**Coste unitario**
PC Sobremesa 1	478,90 €
PC Sobremesa 2	27,00 €
Portátil	1.502,99 €
Netbook	249,00 €
Router Wifi	15,00 €
Switch	11,86 €

Tabla 2.6: Recursos materiales (Hardware).

Concepto	Coste unitario	Núm. De licencias
MS Project 2010	775,00 €	1
MS Visio 2010	330,00 €	1
Antivirus Nod32	39.95 €	2
Kubuntu Linux	0,00 €	2
Apache Ant	0,00 €	4
NetBeans 6.9.1	0,00 €	1
MySQL Sever 5.1	0,00 €	1
MySQL GUI Tools 1.2.17	0,00 €	1
IPTools 1.98	0,00 €	1
TeXnicCenter 1.0	0,00 €	1
Ghostscript 8.71	0,00 €	1
GWView 4.9	0,00 €	1
MiKTeX	0,00 €	1
Gimp 2.6	0,00 €	1
Biblioteca Swing	0,00 €	1
Biblioteca Jasypt	0,00 €	1
JPPF Framework	0,00 €	1
Hazelcast Framework	0,00 €	1

Tabla 2.7: Recursos materiales (Software).

2.3.3. Costo de recursos de trabajo

La tabla 2.8 muestra el costo de los recursos de trabajo.

Concepto	Trabajo	Coste	Importe
Desarrollador	888 h	5,30 €/h	4706,4 €
Total			**4706,4 €**

Tabla 2.8: Costo de recursos de trabajo.

2.3.4. Costo de de recursos materiales

La tabla 2.9 muestra el costo de los recursos materiales.

Concepto	Unidades	Precio	Importe
PC Sobremesa 1	4	1,50 €	6,00 €
PC Sobremesa 2	4	1,50 €	6,00 €
Portátil	25	1,50 €	37,50 €
Netbook	4	1,50 €	6,00 €
Router Wifi	4	0,30 €	1,20 €
Switch	4	0,30 €	1,20 €
Total			**57,90 €**

Tabla 2.9: Costo de recursos materiales.

2.3.5. Amortizaciones

Para el cálculo de las amortizaciones (*Véase la tabla 2.10*) se ha considerado un tiempo de amortización de 3 años . Teniendo en cuenta que el número de horas laborables al año es de 200, el tiempo de amortización (en horas) se obtiene de la siguiente forma: **4800 horas = 3 años x 200 días laborables x 8 horas**

Concepto	Coste unitario	Tiempo de amortización	Coste unitario de amortización	Timpo de uso	Importe
PC Sobremesa 1	478,90 €	4800	0,09977 €	50 h	4,99 €
PC Sobremesa 2	27,00 €	4800	0,00563 €	50 h	0,28 €
Portátil	1.502,99 €	4800	0,31312 €	632 h	197,89 €
Netbook	249,00 €	4800	0,05188 €	50 h	2,59 €
Router Wifi	15,00 €	4800	0,00313 €	48 h	0,15 €
Switch	11,86 €	4800	0,00247 €	48 h	0,12 €
MS Project 2010	775,00 €	4800	0,16146 €	15 h	2,42 €
MS Visio 2010	330,00 €	4800	0,06875 €	6 h	0,41 €
Antivirus Nod32	39.95 €	4800	0,00832 €	665 h	5,53 €
Apache Ant	0,00 €	4800	0,00000 €	378 h	0,00 €
NetBeans 6.9.1	0,00 €	4800	0,00000 €	514 h	0,00 €
MySQL Sever 5.1	0,00 €	4800	0,00000 €	489 h	0,00 €
MySQL GUI Tools 1.2.17	0,00 €	4800	0,00000 €	36 h	0,00 €
IPTools 1.98	0,00 €	4800	0,00000 €	8 h	0,00 €
TeXnicCenter 1.0	0,00 €	4800	0,00000 €	47 h	0,00 €
Ghostscript 8.71	0,00 €	4800	0,00000 €	12 h	0,00 €
GWView 4.9	0,00 €	4800	0,00000 €	12 h	0,00 €
MiKTeX	0,00 €	4800	0,00000 €	47 h	0,00 €
Gimp 2.6	0,00 €	4800	0,00000 €	16 h	0,00 €
Biblioteca Swing	0,00 €	4800	0,00000 €	103 h	0,00 €
Biblioteca Jasypt	0,00 €	4800	0,00000 €	24 h	0,00 €
JPPF Framework	0,00 €	4800	0,00000 €	162 h	0,00 €
Hazelcast Framework	0,00 €	4800	0,00000 €	48 h	0,00 €
Total					**214,38 €**

Tabla 2.10: Amortizaciones

2.3.6. Presupuesto

En la tabla 2.11 se recogen los resultados de las tablas 2.5 a 2.10 a los que se les añade un 10 % en concepto de gastos generales y un 15 % extra en concepto de beneficios para la empresa. Al total obtenido se le suma el 18 % de IVA.

Concepto	**Importe**
Recursos de trabajo	4706,4 €
Recursos materiales	57,90 €
Amortizaciones	214,38 €
Total	**4.978,68 €**
Gastos Generales (10 %)	497,86 €
Beneficio (15 %)	746,81 €
Subtotal	**6.223,35 €**
IVA (18 %)	1.120,21 €
Total	**7.343,56**

Tabla 2.11: Presupuesto

Por tanto, el presupuesto asciende a siete mil trescientos cuarenta y tres euros, con cincuenta y seis céntimos.

Estimando que los datos aportados en el presente presupuesto son suficientes, se somete el proyecto a su aprobación por los organismos oficiales correspondientes.

Firma: Unai Gómez Velasco

Vitoria-Gasteiz, a 22 de Septiembre de 2010

Parte II

Conceptos generales, tecnologías y herramientas

CAPÍTULO

3 Conceptos generales

Este capítulo recoge información elemental sobre la computación en *grid*, los sistemas distribuidos, matemáticas básicas, protocoles de comunicación y criptografía. La finalidad de este capítulo es tener una información general de los conceptos utilizados en la aplicación DJ HACK.

3.1. Clúster

El término **clúster** se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora.

Hoy en día desempeñan un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno.

La tecnología de clústers ha evolucionado en apoyo de actividades que van desde aplicaciones de supercómputo y software de misiones críticas, servidores web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

El cómputo con clústers surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran.

Simplemente, un clúster es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio.

Los clústers son usualmente empleados para mejorar el rendimiento y/o la disponibilidad por encima de la que es provista por un solo computador típicamente siendo más económico que computadores individuales de rapidez y disponibilidad comparables.

La construcción de los ordenadores del clúster es más fácil y económica debido a su flexibilidad: pueden tener todos la misma configuración de hardware y sistema operativo (*clúster homogéneo*), diferente rendimiento pero con arquitecturas y sistemas operativos similares (*clúster semi-homogéneo*), o tener diferente hardware y sistema operativo (*clúster heterogéneo*), lo que hace más fácil y económica su construcción.

Para que un clúster funcione como tal, no basta solo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de manejo del clúster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento.

3.1.1. Beneficios de la tecnología clúster

Las aplicaciones paralelas escalables requieren: buen rendimiento, baja latencia, comunicaciones que dispongan de gran ancho de banda, redes escalables y acceso rápido a archivos. Un clúster puede satisfacer estos requerimientos usando los recursos que tiene asociados a él.

Los clústers ofrecen las siguientes características a un costo relativamente bajo:

- Alto rendimiento
- Alta disponibilidad
- Alta eficiencia
- Escalabilidad

La tecnología clúster permite a las organizaciones incrementar su capacidad de procesamiento usando tecnología estándar, tanto en componentes de hardware como de software que pueden adquirirse a un costo relativamente bajo.

3.1.2. Clasificación de los clústers

El término clúster tiene diferentes connotaciones para diferentes grupos de personas. Los tipos de clústers, establecidos de acuerdo con el uso que se dé y los servicios que ofrecen, determinan el significado del término para el grupo que lo utiliza. Los clústers pueden clasificarse según sus características: se pueden tener clústers de alto rendimiento HPCC *(High Performance Computing clústers)*, clústers de alta disponibilidad HA *(High Availability)* o clústers de alta eficiencia HT (*High Throughput*).

Alto rendimiento: Son clústers en los cuales se ejecutan tareas que requieren de gran capacidad computacional, grandes cantidades de memoria, o ambos a la vez. El llevar a cabo estas tareas puede comprometer los recursos del clúster por largos periodos de tiempo.

Alta disponibilidad: Son clústers cuyo objetivo de diseño es el de proveer disponibilidad y confiabilidad. Estos clústers tratan de brindar la máxima disponibilidad de los servicios que ofrecen. La confiabilidad se provee mediante software que detecta fallos y permite recuperarse frente a los mismos, mientras que en hardware se evita tener un único punto de fallos.

Alta eficiencia: Son clústers cuyo objetivo de diseño es el ejecutar la mayor cantidad de tareas en el menor tiempo posible. Existe independencia de datos entre las tareas individuales. El retardo entre los nodos del clúster no es considerado un gran problema.

Los clústers pueden también clasificar como *clústers de IT Comerciales* (Alta disponibilidad, Alta eficiencia) y *clústers Científicos* (Alto rendimiento). A pesar de las discrepancias a nivel de requerimientos de las aplicaciones, muchas de las características de las arquitecturas de hardware y software, que están por debajo de las aplicaciones en todos estos clústers, son las mismas. Más aún, un clúster de determinado tipo, puede también presentar características de los otros.

3.1.3. Computación en grid

La computación **grid** es una tecnología innovadora que permite utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado. En este sentido es una nueva forma de computación

distribuida, en la cual los recursos pueden ser heterogéneos (diferentes arquitecturas, supercomputadores, clústers...) y se encuentran conectados mediante redes de área extensa (por ejemplo Internet). Desarrollado en ámbitos científicos a principios de los años 1990, su entrada al mercado comercial siguiendo la idea de la llamada *Utility computing* supone una revolución que dará mucho que hablar.

El término grid se refiere a una infraestructura que permite la integración y el uso colectivo de ordenadores de alto rendimiento, redes y bases de datos que son propiedad y están administrados por diferentes instituciones. Puesto que la colaboración entre instituciones envuelve un intercambio de datos, o de tiempo de computación, el propósito del grid es facilitar la integración de recursos computacionales. Universidades, laboratorios de investigación o empresas se asocian para formar grid para lo cual utilizan algún tipo de software que implemente este concepto.

Llamamos grid al sistema de computación distribuido que permite compartir recursos no centrados geográficamente para resolver problemas de gran escala. Los recursos compartidos pueden ser ordenadores (PC, estaciones de trabajo, supercomputadoras, PDA, portátiles, móviles, etc), software, datos e información, instrumentos especiales (radio, telescopios, etc.) o personas/colaboradores.

La computación grid ofrece muchas ventajas frente a otras tecnologías alternativas. La potencia que ofrecen multitud de computadores conectados en red usando grid es prácticamente ilimitada, además de que ofrece una perfecta integración de sistemas y dispositivos heterogéneos, por lo que las conexiones entre diferentes máquinas no generarán ningún problema. Se trata de una solución altamente escalable, potente y flexible, ya que evitarán problemas de falta de recursos (*cuellos de botella*) y nunca queda obsoleta, debido a la posibilidad de modificar el número y características de sus componentes.

Estos recursos se distribuyen en la red de forma transparente pero guardando unas pautas de seguridad y políticas de gestión de carácter tanto técnico como económico. Así pues, su objetivo será el de compartir una serie de recursos en la red de manera uniforme, segura, transparente, eficiente y fiable, ofreciendo un único punto de acceso a un conjunto de recursos distribuidos geográficamente en diferentes dominios de administración. Esto nos puede llevar a pensar que la computación grid permite la creación de empresas virtuales. Es importante saber que una grid es un conjunto de máquinas distribuidas que ayudan a mejorar el trabajo sobre software pesados.

3.2. Computación distribuida

La **computación distribuida** o **informática en malla**, es un nuevo modelo para resolver problemas de computación masiva utilizando un gran número de computadoras organizadas en

racimos incrustados en una infraestructura de telecomunicaciones distribuida.

3.2.1. Sistemas distribuidos

Un **sistema distribuido** se define como una colección de computadoras separados físicamente y conectados entre sí por una red de comunicaciones distribuida; cada máquina posee sus componentes de hardware y software que el usuario percibe como un solo sistema (no necesita saber qué cosas están en qué máquinas). El usuario accede a los recursos remotos de la misma manera en que accede a recursos locales, o un grupo de computadores que usan un software para conseguir un objetivo en común.

Los sistemas distribuidos deben ser muy confiables, ya que si un componente del sistema se descompone otro componente debe de ser capaz de reemplazarlo, esto se denomina *tolerancia a fallos*.

El tamaño de un sistema distribuido puede ser muy variado, ya sean decenas de hosts (red de área local), centenas de hosts (red de área metropolitana), y miles o millones de hosts (Internet); esto se denomina *escalabilidad*.

3.2.2. Características

A continuación se muestran las características principales de un sistema distribuido:

1. Para cada uno de los usuarios debe de ser similar al trabajo en el sistema centralizado.
2. Seguridad interna en el sistema distribuido.
3. Se ejecuta en múltiples computadoras.
4. Tiene varias copias del mismo sistema operativo o de diferentes sistemas operativos que proveen los mismos servicios.
5. Entorno de trabajo cómodo.
6. Dependiente de redes (LAN, MAN, WAN, etc.).
7. Compatibilidad entre los dispositivos conectados.
8. Transparencia (El uso de múltiples procesadores y el acceso remoto debe de ser invisible).

3.2.3. Objetivo

La computación distribuida ha sido diseñada para resolver problemas demasiado grandes para cualquier supercomputadora y mainframe, mientras se mantiene la flexibilidad de trabajar

en múltiples problemas más pequeños. Por lo tanto, la computación en grid es naturalmente un entorno multiusuario; por ello, las técnicas de autorización segura son esenciales antes de permitir que los recursos informáticos sean controlados por usuarios remotos. La historia de la computación puede remontarse a cientos de años atrás, cuando se creaban máquinas para ayudar en tareas de cálculos, como el ábaco. La primera calculadora mecánica fue creada en 1623 por *Wilhelm Schickard*, y *Charles Babbage* diseñó la máquina diferencial en la época victoriana. Eran máquinas que se limitaban a realizar una sola tarea, o como mucho, algún subconjunto de todas las posibles tareas.

Las nuevas y poderosas computadoras comenzaron a ser desarrolladas durante la década del 40, que es también cuando comenzó a hacerse evidente que las computadoras podían usarse para mucho más que simples cálculos matemáticos.

3.3. Criptografía

La **criptografía** (del griego *krypto*, "*oculto*", y *graphos*, "*escribir*", literalmente "*escritura oculta*") es la técnica (bien sea aplicada al arte o la ciencia) que altera las representaciones lingüísticas de un mensaje.

3.3.1. Objetivo

En esencia la criptografía trata de enmascarar las representaciones caligráficas de una lengua, de forma discreta. Si bien, el área de estudio científico que se encarga de ello es la *criptología*.

Para ello existen distintos métodos, en donde el más común es el cifrado. Esta técnica enmascara las referencias originales de la lengua por un método de conversión gobernado por un algoritmo que permita el proceso inverso o descifrado de la información. El uso de esta u otras técnicas, permite un intercambio de mensajes que sólo puedan ser leídos por los destinatarios designados como *coherentes*. Un destinatario coherente es la persona a la que el mensaje se le dirige con intención por parte del remitente. Así pues, el destinatario coherente conoce el discretismo usado para el enmascaramiento del mensaje. Por lo que, o bien posee los medios para someter el mensaje criptográfico al proceso inverso, o puede razonar e inferir el proceso que lo convierta en un mensaje de acceso público. En ambos casos, no necesita usar técnicas criptoanalíticas.

Un ejemplo cotidiano de criptografía es el que usamos cuando mandamos una carta. El mensaje origen queda enmascarado por una cubierta denominada sobre, la cual declara el destinatario coherente, que además conoce el proceso inverso para hacer público el mensaje contenido en el sobre.

Hay procesos más elaborados que, por decirlo de alguna manera, el mensaje origen trata de introducir cada letra usada en un '*sobre*' distinto. Por ejemplo, la frase '*texto de prueba*', pudiera estar representada por la siguiente notación cifrada: CF, F0, 114, 10E, 106, 72, F3, F6, 75, 10C, 111, 118, FB, F6, F5. El '*sobre*' usado es de notación hexadecimal, si bien, el cálculo hexadecimal es de acceso público, no se puede decir que sea un mensaje discreto, ahora, si el resultado de la notación hexadecimal (como es el caso para el ejemplo) es consecuencia de la aplicación de un '*método*' de cierre del '*sobre*' (como lo es la cola de sellado, o el lacre en las tradicionales cartas), el destinatario debe de conocer la forma de abrirlo sin deteriorar el mensaje origen. En otras palabras, debe de conocer la contraseña.

3.3.2. Conceptos

En la jerga de la criptografía, la información original que debe protegerse se denomina *texto en claro* o *texto plano*. El cifrado es el proceso de convertir el texto plano en un galimatías ilegible, denominado *texto cifrado* o *criptograma*. Por lo general, la aplicación concreta del algoritmo de cifrado (también llamado *cifra*) se basa en la existencia de una clave: información secreta que adapta el algoritmo de cifrado para cada uso distinto. Cifra es una antigua palabra arábiga para designar el número cero; en la Antigüedad, cuando Europa empezaba a cambiar del sistema de numeración romano al arábigo, se desconocía el cero, por lo que este resultaba misterioso, de ahí probablemente que cifrado signifique misterioso.

Las dos técnicas más sencillas de cifrado, en la criptografía clásica, son la *sustitución* (que supone el cambio de significado de los elementos básicos del mensaje: las letras, los dígitos o los símbolos) y la *transposición* (que supone una reordenación de los mismos); la gran mayoría de las cifras clásicas son combinaciones de estas dos operaciones básicas.

El descifrado es el proceso inverso que recupera el texto plano a partir del criptograma y la clave. El *protocolo criptográfico* especifica los detalles de cómo se utilizan los algoritmos y las claves (y otras operaciones primitivas) para conseguir el efecto deseado. El conjunto de protocolos, algoritmos de cifrado, procesos de gestión de claves y actuaciones de los usuarios, es lo que constituyen en conjunto un criptosistema, que es con lo que el usuario final trabaja e interactúa.

Existen dos grandes grupos de cifras: los algoritmos que usan una única clave tanto en el proceso de cifrado como en el de descifrado, y los que emplean una clave para cifrar mensajes y una clave distinta para descifrarlos. Los primeros se denominan *cifras simétricas*, de *clave simétrica* o de *clave privada*, y son la base de los algoritmos de cifrado clásico. Los segundos se denominan *cifras asimétricas*, de *clave asimétrica* o de *clave pública* y forman el núcleo de las técnicas de cifrado modernas.

En el lenguaje cotidiano, la palabra código se usa de forma indistinta con cifra. En la jerga de la criptografía, sin embargo, el término tiene un uso técnico especializado: los códigos son un

método de criptografía clásica que consiste en sustituir unidades textuales más o menos largas o complejas, habitualmente palabras o frases, para ocultar el mensaje; por ejemplo, *cielo azul* podría significar *atacar al amanecer*. Por el contrario, las cifras clásicas normalmente sustituyen o reordenan los elementos básicos del mensaje: letras, dígitos o símbolos; en el ejemplo anterior, *rcnm arcteeaal aaa* sería un criptograma obtenido por *transposición*. Cuando se usa una técnica de códigos, la información secreta suele recopilarse en un *libro de códigos*.

Con frecuencia los procesos de cifrado y descifrado se encuentran en la literatura como encriptado y desencriptado, aunque ambos son neologismos erróneos anglicismos de los términos ingleses *encrypt* y *decrypt* todavía sin reconocimiento académico. Hay quien hace distinción entre cifrado/descifrado y encriptado/decriptado según estén hablando de criptografía simétrica o asimétrica, pero la realidad es que la mayoría de los expertos hispanohablantes prefieren evitar ambos neologismos hasta el punto de que el uso de los mismos llega incluso a discernir a los aficionados y novatos en la materia de aquellos que han adquirido más experiencia y profundidad en la misma.

Ideológicamente cifrar equivale a escribir y descifrar a leer lo escrito.

3.3.3. Historia de la criptografía

La historia de la criptografía es larga y abunda en anécdotas. Ya las primeras civilizaciones desarrollaron técnicas para enviar mensajes durante las campañas militares, de forma que si el mensajero era interceptado la información que portaba no corriera el peligro de caer en manos del enemigo. Posiblemente, el primer criptosistema que se conoce fuera documentado por el historiador griego *Polibio*: un sistema de sustitución basado en la posición de las letras en una tabla. También los romanos utilizaron sistemas de sustitución, siendo el método actualmente conocido como *César*, porque supuestamente *Julio César* lo empleó en sus campañas, uno de los más conocidos en la literatura (según algunos autores, en realidad *Julio César* no usaba este sistema de sustitución, pero la atribución tiene tanto arraigo que el nombre de este método de sustitución ha quedado para los anales de la historia). Otro de los métodos criptográficos utilizados por los griegos fue la *escítala espartana*, un método de trasposición basado en un cilindro que servía como clave en el que se enrollaba el mensaje para poder cifrar y descifrar.

En 1465 el italiano *Leon Battista Alberti* inventó un nuevo sistema de sustitución polialfabética que supuso un gran avance de la época. Otro de los criptógrafos más importantes del siglo XVI fue el francés *Blaise de Vigenère* que escribió un importante tratado sobre *la escritura secreta* y que diseñó una cifra que ha llegado a nuestros días asociada a su nombre. A *Selenus* se le debe la obra criptográfica *Cryptomenytices et Cryptographiae* (Luneburgo, 1624). Durante los siglos XVII, XVIII y XIX, el interés de los monarcas por la criptografía fue notable. Las tropas de *Felipe II* emplearon durante mucho tiempo una cifra con un alfabeto de más de 500 símbolos que los matemáticos del rey consideraban inexpugnable. Cuando el matemático

francés *François Viète* consiguió criptoanalizar aquel sistema para el rey de Francia, a la sazón *Enrique IV*, el conocimiento mostrado por el rey francés impulsó una queja de la corte española ante del papa *Pío V* acusando a *Enrique IV* de utilizar magia negra para vencer a sus ejércitos. Por su parte, la reina *María Estuardo*, reina de Escocia, fue ejecutada por su prima *Isabel I de Inglaterra* al descubrirse un complot de aquella tras un criptoanálisis exitoso por parte de los matemáticos de Isabel.

Durante la Primera Guerra Mundial, los alemanes usaron el cifrado *ADFGVX*. Este método de cifrado es similar a la del tablero de ajedrez *Polibio*. Consistía en una matriz de 6 x 6 utilizado para sustituir cualquier letra del alfabeto y los números 0 a 9 con un par de letras que consiste de A, D, F, G, V, o X.

Desde el siglo XIX y hasta la Segunda Guerra Mundial, las figuras más importantes fueron la del holandés *Auguste Kerckhoffs* y la del prusiano *Friedrich Kasiski*. Pero es en el siglo XX cuando la historia de la criptografía vuelve a experimentar importantes avances. En especial durante las dos contiendas bélicas que marcaron al siglo: la Gran Guerra y la Segunda Guerra Mundial. A partir del siglo XX, la criptografía usa una nueva herramienta que permitirá conseguir mejores y más seguras cifras: las máquinas de cálculo. La más conocida de las máquinas de cifrado posiblemente sea la máquina alemana *Enigma*: una máquina de rotores que automatizaba considerablemente los cálculos que era necesario realizar para las operaciones de cifrado y descifrado de mensajes. Para vencer al ingenio alemán, fue necesario el concurso de los mejores matemáticos de la época y un gran esfuerzo computacional. No en vano, los mayores avances tanto en el campo de la criptografía como en el del criptoanálisis no empezaron hasta entonces.

Tras la conclusión de la Segunda Guerra Mundial, la criptografía tiene un desarrollo teórico importante, siendo *Claude Shannon* y sus investigaciones sobre teoría de la información esenciales hitos en dicho desarrollo. Además, los avances en computación automática suponen tanto una amenaza para los sistemas existentes como una oportunidad para el desarrollo de nuevos sistemas. A mediados de los años 70, el *Departamento de Normas y Estándares norteamericano* publica el primer diseño lógico de un cifrador que estaría llamado a ser el principal sistema criptográfico de finales de siglo: el *Estándar de Cifrado de Datos* o *DES*. En esas mismas fechas ya se empezaba a gestar lo que sería la, hasta ahora, última revolución de la criptografía teórica y práctica: los sistemas asimétricos. Estos sistemas supusieron un salto cualitativo importante, ya que permitieron introducir la criptografía en otros campos que hoy día son esenciales, como el de la firma digital.

3.3.4. Criptografía simétrica

La **criptografía simétrica** es un método criptográfico en el cual se usa una misma clave para cifrar y descifrar mensajes. Las dos partes que se comunican han de ponerse de acuerdo de antemano sobre la clave a usar. Una vez ambas tienen acceso a esta clave, el remitente cifra un

mensaje usándola, lo envía al destinatario, y éste lo descifra con la misma.

3.3.4.1. Seguridad

Un buen sistema de cifrado pone toda la seguridad en la clave y ninguna en el algoritmo. En otras palabras, *no debería ser de ninguna ayuda para un atacante conocer el algoritmo que se está usando*. Sólo si el atacante obtuviera la clave, le serviría conocer el algoritmo. Los algoritmos de cifrado ampliamente utilizados tienen estas propiedades(por ejemplo: *GnuPG* en sistemas *GNU*).

Dado que toda la seguridad está en la clave, es importante que sea muy difícil adivinar el tipo de clave. Esto quiere decir que el abanico de claves posibles, o sea, el espacio de posibilidades de claves, debe ser amplio. *Richard Feynman* fue famoso en *Los Álamos* por su habilidad para abrir cajas de seguridad; para alimentar la leyenda que había en torno a él, llevaba encima un juego de herramientas que incluían un estetoscopio. En realidad, utilizaba una gran variedad de trucos para reducir a un pequeño número la cantidad de combinaciones que debía probar, y a partir de ahí simplemente probaba hasta que adivinaba la combinación correcta. En otras palabras, reducía el tamaño de posibilidades de claves.

Actualmente, los ordenadores pueden descifrar claves con extrema rapidez, y ésta es la razón por la cual el tamaño de la clave es importante en los criptosistemas modernos. El algoritmo de cifrado *DES* usa una clave de 56 bits, lo que significa que hay 2^{56} claves posibles (72,057,594,037,927,936 claves). Esto representa un número muy alto de claves, pero un ordenador genérico puede comprobar el conjunto posible de claves en cuestión de días. Una máquina especializada puede hacerlo en horas. Algoritmos de cifrado de diseño más reciente como *3DES*, *Blowfish* e *IDEA* usan claves de 128 bits, lo que significa que existen 2^{128} claves posibles. Esto equivale a muchísimas más claves, y aun en el caso de que todas las máquinas del planeta estuvieran cooperando, tardarían más tiempo en encontrar la clave que la edad del universo.

3.3.4.2. Inconvenientes

El principal problema con los sistemas de cifrado simétrico no está ligado a su seguridad, sino al intercambio de claves. Una vez que el remitente y el destinatario hayan intercambiado las claves pueden usarlas para comunicarse con seguridad, pero ¿qué canal de comunicación que sea seguro han usado para transmitirse las claves? Sería mucho más fácil para un atacante intentar interceptar una clave que probar las posibles combinaciones del espacio de claves.

Otro problema es el número de claves que se necesitan. Si tenemos un número n de personas que necesitan comunicarse entre sí, se necesitan $n/2$ claves para cada pareja de personas que tengan que comunicarse de modo privado. Esto puede funcionar con un grupo reducido de

personas, pero sería imposible llevarlo a cabo con grupos más grandes.

3.3.4.3. Algoritmo RSA

En criptografía, **RSA** (*Rivest, Shamir y Adleman*) es un sistema criptográfico de clave pública desarrollado en 1977. En la actualidad, RSA es el primer y más utilizado algoritmo de este tipo y es válido tanto para cifrar como para firmar digitalmente.

La seguridad de este algoritmo radica en el problema de la factorización de números enteros. Los mensajes enviados se representan mediante números, y el funcionamiento se basa en el producto, conocido, de dos números primos grandes elegidos al azar y mantenidos en secreto. Actualmente estos primos son del orden de 10^{200}, y se prevé que su tamaño aumente con el aumento de la capacidad de cálculo de los ordenadores.

Como en todo sistema de clave pública, cada usuario posee dos claves de cifrado: una pública y otra privada. Cuando se quiere enviar un mensaje, el emisor busca la clave pública del receptor, cifra su mensaje con esa clave, y una vez que el mensaje cifrado llega al receptor, este se ocupa de descifrarlo usando su clave privada.

Se cree que RSA será seguro mientras no se conozcan formas rápidas de descomponer un número grande en producto de primos. La computación cuántica podría proveer de una solución a este problema de factorización.

3.3.4.4. Algoritmo DES

Data Encryption Standard (DES) es un algoritmo de cifrado, es decir, un método para cifrar información, escogido como *FIPS* en los *Estados Unidos* en 1976, y cuyo uso se ha propagado ampliamente por todo el mundo. El algoritmo fue controvertido al principio, con algunos elementos de diseño clasificados, una longitud de clave relativamente corta, y las continuas sospechas sobre la existencia de alguna puerta trasera para la *National Security Agency* (NSA). Posteriormente DES fue sometido a un intenso análisis académico y motivó el concepto moderno del cifrado por bloques y su criptoanálisis.

Hoy en día, DES se considera inseguro para muchas aplicaciones. Esto se debe principalmente a que el tamaño de clave de 56 bits es corto; las claves de DES se han roto en menos de 24 horas. Existen también resultados analíticos que demuestran debilidades teóricas en su cifrado, aunque son inviables en la práctica. Se cree que el algoritmo es seguro en la práctica en su variante de *Triple DES*, aunque existan ataques teóricos.

Desde hace algunos años, el algoritmo ha sido sustituido por el nuevo AES (*Advanced Encryption Standard*).

En algunas ocasiones, DES es denominado también DEA (*Data Encryption Algorithm*).

3.3.5. Criptografía asimétrica

La **criptografía asimétrica** es el método criptográfico que usa un par de claves para el envío de mensajes. Las dos claves pertenecen a la misma persona a la que se ha enviado el mensaje. Una clave es pública y se puede entregar a cualquier persona, la otra clave es privada y el propietario debe guardarla de modo que nadie tenga acceso a ella. Además, los métodos criptográficos garantizan que esa pareja de claves sólo se puede generar una vez, de modo que se puede asumir que no es posible que dos personas hayan obtenido casualmente la misma pareja de claves.

Si el remitente usa la clave pública del destinatario para cifrar el mensaje, una vez cifrado, sólo la clave privada del destinatario podrá descifrar este mensaje, ya que es el único que la conoce. Por tanto se logra la confidencialidad del envío del mensaje, nadie salvo el destinatario puede descifrarlo.

Si el propietario del par de claves usa su clave privada para cifrar el mensaje, cualquiera puede descifrarlo utilizando su clave pública. En este caso se consigue por tanto la *identificación* y *autentificación* del remitente, ya que se sabe que sólo pudo haber sido él quien empleó su clave privada (salvo que alguien se la hubiese podido robar). Esta idea es el fundamento de la firma electrónica.

Los *sistemas de cifrado de clave pública* o sistemas de cifrado asimétricos se inventaron con el fin de evitar por completo el problema del intercambio de claves de los sistemas de cifrado simétricos. Con las claves públicas no es necesario que el remitente y el destinatario se pongan de acuerdo en la clave a emplear. Todo lo que se requiere es que, antes de iniciar la comunicación secreta, el remitente consiga una copia de la clave pública del destinatario. Es más, esa misma clave pública puede ser usada por cualquiera que desee comunicarse con su propietario. Por tanto, se necesitarán sólo n pares de claves por cada n personas que deseen comunicarse entre sí.

3.3.5.1. Bases

Los sistemas de cifrado de clave pública se basan en *funciones-trampa* de un solo sentido que aprovechan propiedades particulares, por ejemplo de los números primos. Una función de un solo sentido es aquella cuya computación es fácil, mientras que su inversión resulta extremadamente difícil. Por ejemplo, es fácil multiplicar dos números primos juntos para obtener uno compuesto, pero es difícil factorizar uno compuesto en sus componentes primos. Una función-trampa de un sentido es algo parecido, pero tiene una *trampa*. Esto quiere decir que si se conociera alguna pieza de la información, sería fácil computar el inverso. Por ejemplo, *si tenemos un número compuesto por dos factores primos y conocemos uno de los factores, es fácil computar el segundo.*

Dado un cifrado de clave pública basado en factorización de números primos, la clave pública contiene un número compuesto de dos factores primos grandes, y el algoritmo de cifrado usa ese compuesto para cifrar el mensaje. El algoritmo para descifrar el mensaje requiere el conocimiento de los factores primos, para que el descifrado sea fácil si poseemos la clave privada que contiene uno de los factores, pero extremadamente difícil en caso contrario.

3.3.5.2. Descripción

Las dos principales ramas de la criptografía de clave pública son:

- *Cifrado de clave pública:* Un mensaje cifrado con la clave pública de un destinatario no puede ser descifrado por nadie, excepto un poseedor de la clave privada correspondiente– presumiblemente, este será el propietario de esa clave y la persona asociada con la clave pública utilizada. Se utiliza para confidencialidad.

- *Firmas digitales:* Un mensaje firmado con la clave privada del remitente puede ser verificado por cualquier persona que tenga acceso a la clave pública del remitente, lo que demuestra que el remitente tenía acceso a la clave privada (y por lo tanto, es probable que sea la persona asociada con la clave pública utilizada) y la parte del mensaje que no se ha manipulado. Sobre la cuestión de la autenticidad.

Una analogía con el cifrado de clave pública es la de un buzón con una ranura de correo. La ranura de correo está expuesta y accesible al público; su ubicación (la dirección de la calle) es, en esencia, la clave pública. Alguien que conozca la dirección de la calle puede ir a la puerta y colocar un mensaje escrito a través de la ranura; sin embargo, sólo la persona que posee la clave puede abrir el buzón de correo y leer el mensaje.

Una analogía para firmas digitales es el sellado de un envolvente con un personal, sello de cera. El mensaje puede ser abierto por cualquier persona, pero la presencia del sello autentifica la remitente.

Un problema central para el uso de la criptografía de clave pública es de confianza (idealmente prueba) que una clave pública es correcta, pertenece a la persona o entidad que afirmó (es decir, es *auténtico*) y no ha sido manipulado o reemplazados por un tercero malintencionado. El enfoque habitual a este problema consiste en utilizar una infraestructura de clave pública (*PKI*), en la que una o más terceras partes, conocidas como entidades emisoras de certificados, certifican la propiedad de los pares de claves. Otro enfoque, utilizado por *PGP*, es la *web de confianza*, método para asegurar la autenticidad de pares de clave

3.3.5.3. Seguridad:

Como con los sistemas de cifrado simétricos buenos, con un buen sistema de cifrado de clave pública toda la seguridad descansa en la clave y no en el algoritmo. Por lo tanto, el tamaño de la clave es una medida de la seguridad del sistema, pero no se puede comparar el tamaño de la clave del cifrado simétrico con el del cifrado de clave pública para medir la seguridad. En un *ataque de fuerza bruta* sobre un cifrado simétrico con una clave del tamaño de 80 bits, el atacante debe probar hasta $2^{80} - 1$ claves para encontrar la clave correcta. En un ataque de fuerza bruta sobre un cifrado de clave pública con una clave del tamaño de 512 bits, el atacante debe factorizar un número compuesto codificado en 512 bits (hasta 155 dígitos decimales). La cantidad de trabajo para el atacante será diferente dependiendo del cifrado que esté atacando. Mientras 128 bits son suficientes para cifrados simétricos, dada la tecnología de factorización de hoy en día, se recomienda el uso de claves públicas de 1024 bits para la mayoría de los casos.

3.3.5.4. Ventajas del cifrado asimétrico:

La mayor ventaja de la criptografía asimétrica es que se puede cifrar con una clave y descifrar con la otra, pero este sistema tiene bastantes desventajas:

- Para una misma longitud de clave y mensaje se necesita mayor tiempo de proceso.
- Las claves deben ser de mayor tamaño que las simétricas.
- El mensaje cifrado ocupa más espacio que el original.

3.3.5.5. Algoritmo MD5

En criptografía, **MD5** (abreviatura de *Message-Digest Algorithm 5*, Algoritmo de Resumen del Mensaje 5) es un algoritmo de reducción criptográfico de 128 bits ampliamente usado.

MD5 es uno de los algoritmos de reducción criptográficos diseñados por el profesor *Ronald Rivest* del MIT (*Massachusetts Institute of Technology*, Instituto Tecnológico de Massachusetts). Fue desarrollado en 1991 como reemplazo del algoritmo MD4 después de que *Hans Dobbertin* descubriese su debilidad.

A pesar de su amplia difusión actual, la sucesión de problemas de seguridad detectados desde que, en 1996, *Hans Dobbertin* anunciase una colisión de hash, plantea una serie de dudas acerca de su uso futuro.

La codificación del MD5 de 128 bits es representada típicamente como un número de 32 dígitos hexadecimal. El siguiente código de 28 bytes ASCII será tratado con MD5 y se puede ver su correspondiente hash de salida:

MD5(Esto sí es una prueba de MD5) = e99008846853ff3b725c27315e469fbc

Un simple cambio en el mensaje nos da un cambio total en la codificación hash, en este caso cambiamos dos letras, el *"sí"* por un *"no"*.

MD5(Esto no es una prueba de MD5) = dd21d99a468f3bb52a136ef5beef5034

3.3.5.6. Algoritmo SHA

La familia **SHA** (*Secure Hash Algorithm, Algoritmo de Hash Seguro*) es un sistema de funciones hash criptográficas relacionadas de la *Agencia de Seguridad Nacional de los Estados Unidos* y publicadas por el *National Institute of Standards and Technology* (NIST). El primer miembro de la familia fue publicado en 1993 es oficialmente llamado SHA. Sin embargo, hoy día, no oficialmente se le llama SHA-0 para evitar confusiones con sus sucesores. Dos años más tarde el primer sucesor de SHA fue publicado con el nombre de SHA-1. Existen cuatro variantes más que se han publicado desde entonces cuyas diferencias se basan en un diseño algo modificado y rangos de salida incrementados: SHA-224, SHA-256, SHA-384, y SHA-512 (llamándose SHA-2 a todos ellos).

En 1998, un ataque a SHA-0 fue encontrado pero no fue reconocido para SHA-1, se desconoce si fue la NSA quien lo descubrió pero aumentó la seguridad del SHA-1.

SHA-1 ha sido examinado muy de cerca por la comunidad criptográfica pública, y no se ha encontrado ningún ataque efectivo. No obstante, en el año 2004, un número de ataques significativos fueron divulgados sobre funciones criptográficas de hash con una estructura similar a SHA-1; lo que ha planteado dudas sobre la seguridad a largo plazo de SHA-1.

SHA-0 y SHA-1 producen una salida resumen de 160 bits (20 bytes) de un mensaje que puede tener un tamaño máximo de 2^{64} bits, y se basa en principios similares a los usados por el profesor *Ronald L. Rivest* del *MIT* en el diseño de los algoritmos de resumen de mensaje MD4 y MD5.

A continuación se puede ver el hasheo de dos cadenas de caracteres prácticamente iguales:

SHA-1(Esto no es una prueba de SHA-1) =
33a98fc56072421b8613a9d7dce13d38c2da91f2

SHA-1(Esto si es una prueba de SHA-1) =
7c1e16838a651b9719d8a635027ba57db86ecddf

3.3.6. Criptografía híbrida en SSL

La **criptografía híbrida** es un método criptográfico que usa tanto un cifrado simétrico como un asimétrico. Emplea el cifrado de clave pública para compartir una clave para el cifrado simétrico. El mensaje que se esté enviando en el momento, se cifra usando la clave y enviándolo al destinatario. Ya que compartir una clave simétrica no es seguro, la clave usada es diferente para cada sesión.

Secure Sockets Layer / Protocolo de Capa de Conexión Segura (SSL) y *Transport Layer Security* / Seguridad de la Capa de Transporte (TLS), su sucesor, son protocolos criptográficos que proporcionan comunicaciones seguras por una red, comúnmente Internet.

Existen pequeñas diferencias entre SSL 3.0 y TLS 1.0, pero el protocolo permanece sustancialmente igual. El término SSL según se usa aquí, se aplica a ambos protocolos a menos que el contexto indique lo contrario.

3.3.6.1. Descripción

SSL proporciona autenticación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía. Habitualmente, sólo el servidor es autenticado (es decir, se garantiza su identidad) mientras que el cliente se mantiene sin autenticar; la autenticación mutua requiere un despliegue de infraestructura de claves públicas (o PKI) para los clientes. Los protocolos permiten a las aplicaciones cliente-servidor comunicarse de una forma diseñada para prevenir escuchas (*eavesdropping*), la falsificación de la identidad del remitente (*phishing*) y alterar la integridad del mensaje.

SSL implica una serie de fases básicas:

- Negociar entre las partes el algoritmo que se usará en la comunicación.
- Intercambio de claves públicas y autenticación basada en certificados digitales.
- Cifrado del tráfico basado en cifrado simétrico.

Durante la primera fase, el cliente y el servidor negocian qué algoritmos criptográficos se van a usar. Las implementaciones actuales proporcionan las siguientes opciones:

- Para criptografía de clave pública: RSA, Diffie-Hellman, DSA (*Digital Signature Algorithm*) o Fortezza.

- Para cifrado simétrico: RC2, RC4, IDEA (*International Data Encryption Algorithm*), DES (*Data Encryption Standard*), Triple DES o AES (*Advanced Encryption Standard*)
- Con funciones hash: MD5 o de la familia SHA.

3.3.6.2. Funcionamiento

El protocolo SSL intercambia registros; opcionalmente, cada registro puede ser comprimido, cifrado y empaquetado con un código de autentificación del mensaje (MAC). Cada registro tiene un campo de *content_type* que especifica el protocolo de nivel superior que se está usando.

Cuando se inicia la conexión, el nivel de registro encapsula otro protocolo, el protocolo *handshake*, que tiene el *content_type 22*.

El cliente envía y recibe varias estructuras *handshake*:

- Envía un mensaje *ClientHello* especificando una lista de conjunto de cifrados, métodos de compresión y la versión del protocolo SSL más alta permitida. Éste también envía bytes aleatorios que serán usados más tarde (llamados *Challenge de Cliente o Reto*). Además puede incluir el identificador de la sesión.
- Después, recibe un registro *ServerHello*, en el que el servidor elige los parámetros de conexión a partir de las opciones ofertadas con anterioridad por el cliente.
- Cuando los parámetros de la conexión son conocidos, cliente y servidor intercambian certificados (dependiendo de las claves públicas de cifrado seleccionadas). Estos certificados son actualmente *X.509*, pero hay también un borrador especificando el uso de certificados basados en *OpenPGP*.
- El servidor puede requerir un certificado al cliente, para que la conexión sea mutuamente autentificada.
- Cliente y servidor negocian una clave secreta (simétrica) común llamada *master secret*, posiblemente usando el resultado de un intercambio *Diffie-Hellman*, o simplemente cifrando una clave secreta con una clave pública que es descifrada con la clave privada de cada uno. Todos los datos de claves restantes son derivados a partir de este *master secret* (y los valores aleatorios generados en el cliente y el servidor), que son pasados a través una función pseudoaleatoria cuidadosamente elegida.

3.4. Protocolos TCP/IP

Protocolo de Control de Transmisión (TCP) y **Protocolo de Internet** (IP), fueron los dos primeros en definirse, y son los más utilizados de la familia de protocolos de Internet.

Existen tantos protocolos en este conjunto que llegan a ser más de 100 diferentes, entre ellos se encuentra el popular HTTP (*HyperText Transfer Protocol*), que es el que se utiliza para acceder a las páginas web, además de otros como el ARP (*Address Resolution Protocol*) para la resolución de direcciones, el FTP (*File Transfer Protocol*) para transferencia de archivos, y el SMTP (*Simple Mail Transfer Protocol*) y el POP (*Post Office Protocol*) para correo electrónico, *TELNET* para acceder a equipos remotos, entre otros.

El TCP/IP es la base de Internet, y sirve para enlazar computadoras que utilizan diferentes sistemas operativos, incluyendo PC, minicomputadoras y computadoras centrales sobre redes de área local (LAN) y área extensa (WAN). TCP/IP fue desarrollado y demostrado por primera vez en 1972 por el *Departamento de Defensa de los Estados Unidos*, ejecutándolo en *ARPANET*, una red de área extensa de dicho departamento.

El conjunto TCP/IP está diseñado para enrutar y tiene un grado muy elevado de fiabilidad, es adecuado para redes grandes y medianas, así como en redes empresariales. Se utiliza a nivel mundial para conectarse a Internet y a los servidores web. Es compatible con las herramientas estándar para analizar el funcionamiento de la red.

Un inconveniente de TCP/IP es que es más difícil de configurar y de mantener que *NetBEUI* o *IPX/SPX*; además es algo más lento en redes con un volumen de tráfico medio bajo. Sin embargo, puede ser más rápido en redes con un volumen de tráfico grande donde haya que enrutar un gran número de tramas.

El conjunto TCP/IP se utiliza tanto en redes empresariales como por ejemplo en campus universitarios o en complejos empresariales, en donde utilizan muchos enrutadores y conexiones a mainframe o a ordenadores *UNIX*, como así también en redes pequeñas o domésticas, y hasta en teléfonos móviles y en domótica.

3.5. Agrupamiento de conexiones

En computación, se denomina **connection pool** (**agrupamiento de conexiones**) al manejo de una colección de conexiones abiertas a una base de datos de manera que puedan ser reutilizadas al realizar múltiples consultas o actualizaciones.

Cada vez que un programa cliente necesita comunicarse con una base de datos, establece una conexión, generalmente utilizando un protocolo especializado. Esta conexión, generalmente se mantiene abierta el tiempo que dura la ejecución del programa y sólo es cerrada al finalizar el trabajo de la aplicación con la base de datos.

Este esquema, sin embargo, no es apropiado para aplicaciones multitarea en las que el mismo programa puede querer realizar en paralelo más de una operación sobre la base de datos. Este caso es típico de las aplicaciones que proveen servicio de páginas web a múltiples usuarios; en ellas, el número de operaciones sobre la base de datos y su cadencia dependen de la

actividad de los usuarios de las páginas servidas.

Al mantenerse abierto un grupo de conexiones, éstas son atribuidas a los diferentes hilos de ejecución únicamente el tiempo de una transacción con la base de datos. Al finalizar su utilización, la conexión se pone a disposición de otro hilo de ejecución que necesite de ese recurso, en lugar de cerrarla o de asignarla permanentemente a un único hilo de ejecución.

Según las políticas de agrupamiento de conexiones, cuando todas están en uso se establecen nuevas conexiones, y si ello no es posible, se deja el hilo de ejecución en espera de la liberación de alguna conexión. A la inversa, si pasa mucho tiempo sin que se utilicen las conexiones, algunas de ellas o todas podrían ser cerradas.

3.6. Matemática básica

En esta sección se muestran los conceptos matemáticos aplicados en DJ HACK, como son las variaciones con repetición, los sistemas de numeración utilizados para realizar los cálculos de palabras que se utilizan en la fuerza bruta[1].

3.6.1. Variaciones con repetición

Si se dispone de n objetos $a_1 \quad a_2 \quad \dots \quad a_n$, se pueden formar grupos ordenados de m de ellos, pudiéndose repetir de muchas maneras:

$$
\begin{array}{cccc}
a_1 & a_1 & \dots & a_1 \\
a_1 & a_1 & \dots & a_2 \\
a_1 & a_1 & \dots & a_3 \\
 & & \dots & \\
a_i & a_i & \dots & a_i \\
 & & \dots & \\
a_n & a_n & \dots & a_{n-2} \\
a_n & a_n & \dots & a_{n-1} \\
a_n & a_n & \dots & a_n
\end{array}
$$

Se determina que estos grupos ordenados son **variaciones con repetición** de estos n elementos de orden m, o también, tomados de m en m.

[1] Véase más información sobre fuerza bruta en el apéndice 5.1

Mientras que con *n* objetos solamente se pueden formar *variaciones ordinarias* de órdenes 1, 2, 3, . . . , n; en cambio, se pueden formar *variaciones con repetición* de cualquier orden, por grande que sea.

El número de variaciones con repetición de *n* elementos tomados de *m* en *m* se denota por VR_m^n , y equivale a:

$$VR_m^n = n \times n \times n \times \ldots \times n = n^m$$

3.6.2. Sistemas de numeración

Un sistema de numeración es un conjunto de símbolos y reglas de generación que permiten construir todos los números válidos.

Un sistema de numeración puede representarse como:

$$\mathcal{N} = (S, \mathcal{R})$$

donde:

- $\mathcal{N}$ es el sistema de numeración considerado (p.ej. decimal, binario, etc.).
- S es el conjunto de símbolos permitidos en el sistema. En el caso del sistema decimal son 0,1,...9; en el binario son 0,1; en el octal son 0,1,. . . ,7; en el hexadecimal son 0,1,. . . ,9,A,B,C,D,E,F.
- $\mathcal{R}$ son las reglas que nos indican qué números son válidos en el sistema, y cuáles no. En un sistema de numeración posicional las reglas son bastante simples, mientras que la numeración romana requiere reglas algo más elaboradas.

Estas reglas son diferentes para cada sistema de numeración considerado, pero una regla común a todos es que para construir números válidos en un sistema de numeración determinado sólo se pueden utilizar los símbolos permitidos en ese sistema.

Para indicar en qué sistema de numeración se representa una cantidad se añade como subíndice a la derecha el número de símbolos que se pueden representar en dicho sistema.

3.6.2.1. Sistemas de numeración posicionales

El número de símbolos permitidos en un sistema de numeración posicional se conoce como base del sistema de numeración. Si un sistema de numeración posicional tiene base *b* significa que disponemos de *b* símbolos diferentes para escribir los números, y que *b* unidades forman una unidad de orden superior.

Ejemplo en el sistema de numeración decimal:

Si contamos desde 0, incrementando una unidad cada vez, al llegar a 9 unidades, hemos agotado los símbolos disponibles, y si queremos seguir contando no disponemos de un nuevo símbolo para representar la cantidad que hemos contado. Por tanto añadimos una nueva columna a la izquierda del número, reutilizamos los símbolos de que disponemos, decimos que tenemos una unidad de segundo orden (decena), ponemos a cero las unidades, y seguimos contando.

De igual forma, cuando contamos hasta 99, hemos agotado los símbolos disponibles para las dos columnas; por tanto si contamos (sumamos) una unidad más, debemos poner a cero la columna de la derecha y sumar 1 a la de la izquierda (decenas). Pero la columna de la izquierda ya ha agotado los símbolos disponibles, así que la ponemos a cero, y sumamos 1 a la siguiente columna (centena). Como resultado nos queda que $99+1=100$.

El cuentakilómetros mecánico, al utilizar el sistema de numeración posicional decimal, nos muestra lo anterior: va sumando 1 a la columna de la derecha y cuando la rueda de esa columna ha completado una vuelta (se agotan los símbolos), se pone a cero y se añade una unidad a la siguiente columna de la izquierda.

Pero estamos tan habituados a contar usando el sistema decimal que no somos conscientes de este comportamiento, y damos por hecho que $99+1=100$, sin pararnos a pensar en el significado que encierra esa expresión.

Tal es la costumbre de calcular en decimal que la mayoría de la población ni siquiera se imagina que puedan existir otros sistemas de numeración diferentes al de base 10, y tan válidos y útiles como este. Entre esos sistemas se encuentran el de base 2 sistema binario, de base 8 sistema octal y el de base 16 sistema hexadecimal. También los antiguos mayas tuvieron un sistema de numeración posicional el cual ya no se usa.

3.6.2.2. Teorema fundamental de la numeración

Este teorema establece la forma general de construir números en un sistema de numeración posicional. Primero estableceremos unas definiciones básicas:

N , número válido en el sistema de numeración.

b , base del sistema de numeración. Número de símbolos permitidos en el sistema.

d_i , un símbolo cualquiera de los permitidos en el sistema de numeración.

n , número de dígitos de la parte entera.

, , coma fraccionaria. Símbolo utilizado para separar la parte entera de un número de su parte fraccionaria.

k , número de dígitos de la parte decimal.

La fórmula general para construir un número N, con un número finito de decimales, en un sistema de numeración posicional de base b es la siguiente:

$$N = \begin{cases} \langle d_{(n-1)} \dots d_1 d_0, d_{-1} \dots d_{-k} \rangle = \sum_{i=-k}^{n} d_i b^i \\ N = d_n b^n + \dots + d_1 b^1 + d_0 b^0 + d_{-1} b^{-1} + \dots + d_{-k} b^{-k} \end{cases}$$

El valor total del número será la suma de cada dígito multiplicado por la potencia de la base correspondiente a la posición que ocupa en el número.

Esta representación posibilita la realización de sencillos algoritmos para la ejecución de operaciones aritméticas.

CAPÍTULO

4 Desarrollo técnico

En este capítulo se describen las tecnologías y herramientas claves utilizadas en la realización del proyecto.

4.1. Estudio de la tecnología

4.1.1. JDK

Java Development Kit o (JDK), es un software que provee herramientas de desarrollo para la creación de programas en *Java*. Puede instalarse en una computadora local o en una unidad de red.

En la unidad de red se pueden tener las herramientas distribuidas en varias

computadoras y trabajar como una sola aplicación.

4.1.1.1. Herramientas

JDK dispone de las siguientes herramientas:

- *Appletviewer*: es un visor de *applet* para generar sus vistas previas, ya que un *applet* carece de método *main* y no se puede ejecutar con el programa java.
- *Javac*: es el compilador de JAVA.
- *Java*: es el intérprete de JAVA.
- *Javadoc*: genera la documentación de las clases java de un programa.

4.1.2. JPPF

Java Parallel Processing Framework (JPPF) permite a las aplicaciones con grandes necesidades de potencia que van a ser ejecutadas en cualquier número de computadoras, reducir dramáticamente su tiempo de procesamiento. Esto se hace mediante el fraccionamiento de una aplicación en partes más pequeñas que pueden ser ejecutadas simultáneamente en diferentes máquinas.

4.1.2.1. Funcionamiento

Existen dos aspectos:

- La división de una aplicación en partes más pequeñas que se pueden ejecutar de forma independiente y en paralelo.

  JPPF ofrece servicios que hacen que este esfuerzo mucho más fácil, más rápido y mucho menos doloroso que sin ellos. El resultado es un objeto JPPF llamado *job* (*Trabajo*), compuesto de partes más pequeñas llamadas *task* (*Tareas*).

- Ejecutar la aplicación en el grid JPPF.

  Un *JPPF grid* está hecho de un servidor, para que cualquier número de nodos de ejecución se conecten a él. Un nodo es un componente de software JPPF que generalmente se instala y se ejecuta en un equipo diferente. Esto comúnmente se llama una arquitectura *maestro/esclavo*, donde se distribuye el trabajo por el servidor (también conocido como *maestro*) a los nodos (también conocidos como *esclavos*). En términos JPPF, a una unidad de trabajo se le llama un *job*, construido por *tareas* que son distribuidas por el servidor entre los nodos de ejecución en paralelo.

4.1.2.2. Ventajas

Entre los beneficios JPPF están su facilidad de instalación, uso y despliegue. No es necesario pasar unos días para escribir un *Hola Mundo.* Un par de minutos, hasta un par de horas como máximo, serán suficientes. La implementación de componentes JPPF sobre un clúster es algo tan simple como copiar fichero sobre FTP o cualquier sistema de archivos de red. JPPF permite a los desarrolladores centrarse en su núcleo de desarrollo de software, en lugar de perder el tiempo en la complejidad del procesamiento paralelo y distribuido.

Con el framework 100 % Java, JPPF se ejecutará en cualquier sistema que soporte Java: MacOS, Windows, Linux, zOS, en cualquier hardware, en computadoras portátiles simples y hasta en una computadora central. Esto no quiere decir que JPPF está limitado a correr trabajos únicamente en Java. Se puede correr cualquier tipo de aplicación, sea de Java o no, en la plataforma como *job* de JPPF.

Otro beneficio de JPPF es una simplificación, casi inmediata, del proceso de despliegue de la aplicación en el grid. Aunque la aplicación se ejecutará en todos los nodos a la vez, sólo es necesario incorporar este sistema en una única ubicación. Al extender de la clase *Java class loading mechanism*, JPPF elimina la mayor parte de la carga de la implementación del ciclo de vida de la aplicación, acortando dramáticamente el tiempo de salida al mercado y el tiempo de producción.

4.1.2.3. Características

- **Características principales:** Hay mucho más para JPPF que correr y desplegar sus aplicaciones en la red. Las características son tan numerosas que en esta sección sólo se ven las principales:

  - *API completa y fácil de usar:* Pasar de un modelo de aplicación de un único subproceso a un modelo paralelo basado en grid puede ser una tarea desalentadora. JPPF facilita este trabajo, proporcionando a los desarrolladores un conjunto de APIs que son simples, se puede aprender rápidamente y requieren una mínima o ninguna modificación del código existente.
  - *No al uso de la configuración:* En la mayoría de entornos, JPPF se puede implementar sin ninguna carga de configuración adicional. Nodos y clientes de la aplicación detectarán automáticamente los servidores en la red. El servidor se adaptará automáticamente a los en carga de trabajo y optimizará el rendimiento. El código y las bibliotecas necesarias serán automáticamente desplegadas donde se necesiten.
  - *Dinámica de la ampliación de la red y la auto-reparación:* El grid de JPPF es tolerante a fallos, lo que significa que el fallo de un nodo, o incluso un servidor, no pone en peligro los *jobs* se está ejecutando actualmente o que están pendientes por

ejecutar. En la mayoría de los casos, la degradación del rendimiento será apenas perceptible, ya que JPPF se adapta automáticamente a la topología y los cambios en la carga de trabajo.

Por otra parte, los nodos y servidores pueden ser iniciados dinámicamente y serán automáticamente reconocidos, permitiendo a JPPF funcionar en el modo *crunch mode*. Además de esto, los tres componentes de JPPF se benefician de las funcionalidades de recuperación automática.

- *Trabajo a nivel de SLA:* Cada *job* presentado a la red JPPF, se ejecuta dentro de los límites definidos por su propio *SLA* (*Service Level Agreement*). Esto permite determinar las características (es decir, la memoria disponible, la carga de los procesadores, espacio en disco, sistemas operativos, etc.) de los nodos que se usaran para computar un trabajo, así como cuántos nodos se usaran en el *job*.
- *Gestión y seguimiento:* Las funciones de administración y seguimiento se descubren a través de la monitorización de los estados de los servidores y nodos, estadísticas detalladas y eventos, administración remota, monitoreo en tiempo real a nivel de trabajo y gestión, los gráficos estadísticos, la carga de CPU... Estas funcionalidades están disponibles a través de una interfaz gráfica de usuario, así como de la API de JPPF.
- *La integración con aplicaciones líderes y servidores web* Al cumplir con la especificación *Java Connector Architecture 1.5*, JPPF se integra perfectamente con la completa oferta de los principales servidores de aplicaciones *J2EE*: *JBoss* ®, *Glassfish* ®, *IBM Websphere* ®, *Oracle WebLogic* ®, *Oracle* ® *OC4J*. JPPF también se integra con la plataforma de aplicaciones *GigaSpaces Xtreme* ® y *Apache Tomcat servidor Web* ®.

- **Características completas:**
  - *Facilidad de uso:*
    - API simple que requiere pequeña o ninguna curva de aprendizaje.
    - Despliegue automático del código de la aplicación en el grid.
    - Capacidad de reutilizar los objetos existentes o heredados sin modificación.
    - *Happy path* sin necesidad de configuración adicional.
    - Descubrimiento automático del servidor.
    - Plantilla de la aplicación conveniente y reusable para un inicio rápido y fácil en el desarrollo de aplicaciones JPPF.
    - Interfaz de servicio directo a la red de JPPF.
  - *Auto-reparación y recuperación:*
    - Reconexión automática de los nodos con la estrategia de conmutación por error.

- Reconexión automática del cliente con la estrategia de conmutación por error.
- La tolerancia a fallos con la reinserción de *jobs* a la cola del servidor.

- *Trabajo a nivel de SLA:*
  - Las políticas de ejecución habilitan el filtrado del nodo basado en reglas.
  - El número máximo de nodos que se pueden ejecutar en cada uno de los *jobs*.
  - Priorización de trabajos.
  - Capacidad de programar la fecha de ejecución de un trabajo determinado.
  - Vencimiento de la fecha planeada de la ejecución de un trabajo.
- *Gestión y seguimiento:*
  - Eventos a nivel de tarea.
  - Eventos a nivel de *job*.
  - Estadísticas del rendimiento del servidor.
  - Gráficos del rendimiento del servidor.
  - Gráficos definidos por el usuario.
  - Control y seguimiento remoto de los servidores.
  - Monitorización del uso de la CPU.
  - Seguimiento del equilibrio de la carga.
  - Gráficos definidos por el usuario.
  - Gestión y seguimiento disponibles a través de la API y la interfaz gráfica de usuario (consola de administración).
- *Plataforma de extensibilidad:*
  - Todos los *beans* de gestión se pueden conectar, los usuarios pueden añadir sus propios módulos de gestión a nivel de servidor o nodo.
  - Clases de inicio: los usuarios pueden añadir sus propios módulos de inicialización en los sistemas de puesta en marcha del servidor y del nodo.
  - Seguridad: Cualquier dato transmitido a través de la red puede ser encriptado mediante las transformaciones de los datos definidas por el usuario.
  - Los módulos de equilibrio de carga permiten a los usuarios escribir sus propias estrategias de equilibrio.
  - Capacidad para especificar esquemas alternativos de serialización.
- *Rendimiento y eficiencia de los recursos:*
  - Múltiples algoritmos configurables de equilibrio de carga.
  - Los cambios en el equilibrio de carga se ajustan en tiempo real y en caliente.
  - Memoria consciente de los servidores y nodos con desbordamiento de disco.
  - Pool de conexiones a los servidores del lado del cliente.

- Memoria consciente de los servidores y nodos con desbordamiento de disco.
- El lado del cliente *pool* de conexiones del servidor.

- *Ampliación dinámica de la topología:*
  - Los nodos se pueden agregar y quitar de forma dinámica de la red.
  - Los servidores se pueden agregar y quitar de forma dinámica de la red.
  - Los servidores pueden trabajar solos o unidos en topología *P2P* con otros servidores.
- *Conectores de tercera generación:*
  - Conector *J2EE* compatible con *JCA 1.5*, desplegado como un adaptador de recursos estándar.
  - Conector *GigaSpaces XAP*.
  - Conector *Apache Tomcat*.
- *Add-ons:*
  - Multiplexor TCP. Posibilidad del envió del tráfico de la red JPPF a través de un único puerto para trabajar con entornos con cortafuegos.
- *Modos de implementación:*
  - Todos los componentes desplegables como aplicaciones Java independientes.
  - Los servidores y nodos se despliegan como demonios *Linux* / *Unix*.
  - Los servidores y nodos se despliegan como los servicios de *Windows*.
  - Despliegue de aplicaciones cliente como web, *J2EE* o aplicaciones *GigaSpaces XAP*.
- *Modos de ejecución:*
  - Propuesta de trabajo síncrona o asíncrona.
  - El cliente puede ejecutar en modo local (beneficioso para los sistemas con muchas CPU).
  - El cliente puede ejecutar en modo distribuido (la ejecución se delega a los nodos remotos).
  - El cliente puede ejecutar en modo mixto local/distribuido con una adaptación de equilibrio de carga.

4.1.2.4. Arquitectura

- **Arquitectura de alto nivel:** En la figura 4.1 se puede ver de forma esquematizada la arquitectura usada por JPPF.

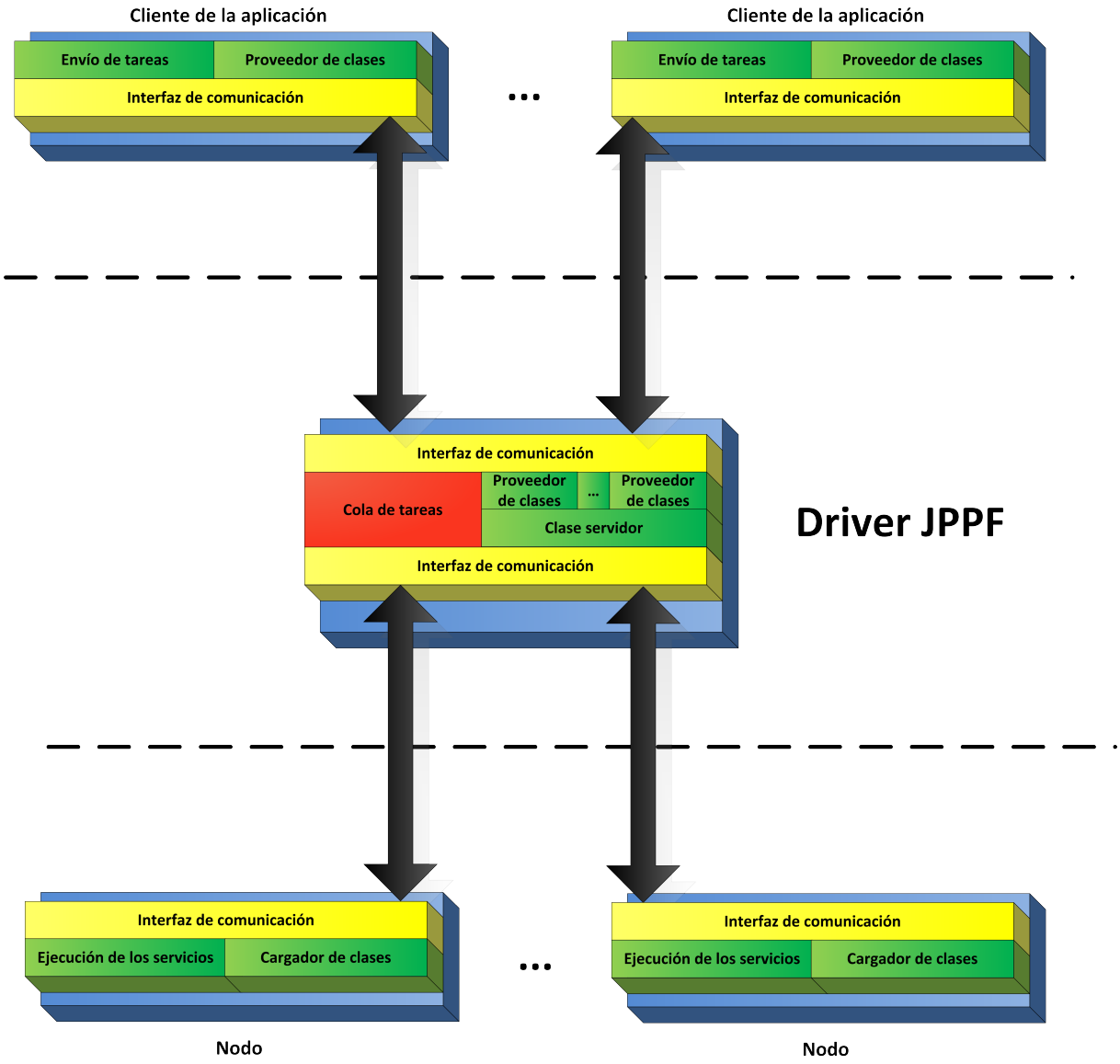


Figura 4.1: Arquitectura de una red en JPPF

- **Descripción de los componentes** A partir del diagrama que se muestra en la figura 4.2, se puede ver que el marco de trabajo tiene una arquitectura de 3 niveles, constando de 3 capas distintas:

  1. *Capa de cliente: Proporciona un API y herramientas de comunicación que utilizan el framework para enviar las tareas a ejecutar en paralelo.*
  2. *Capa de servicio*: Responsable de la comunicación entre los clientes y los nodos, junto con la gestión de la cola de ejecución, el balanceo de carga y características de recuperación, así como la carga dinámica de la clases de la aplicación en los nodos correspondientes.
  3. *Capa de ejecución*: Estos son los nodos. Ellos ejecutan las tareas individuales, devuelven los resultados de la ejecución, y solicitan dinámicamente, desde el servidor

de JPPF, el código necesario para ejecutar las tareas del cliente.

- **Flujo de ejecución:** El esquema que se puede ver en la figura 4.2 representa el flujo de ejecución de una tarea sobre JPPF.

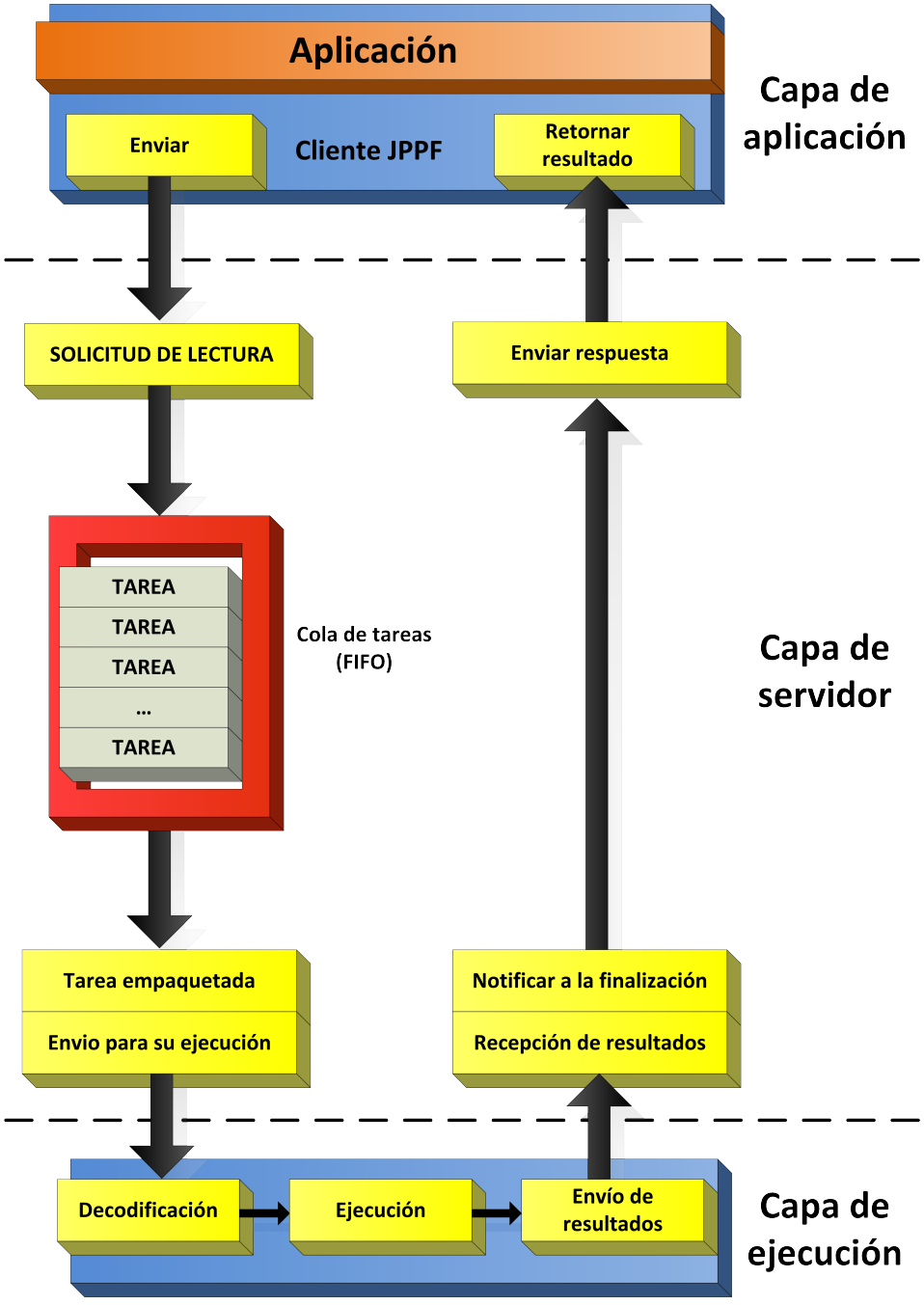


Figura 4.2: Flujo de ejecución de una tarea.

- **Carga de clases en JPPF** En la figura 4.3 se muestra el flujo de despliegue y la carga de las clases en un sistema JPPF.

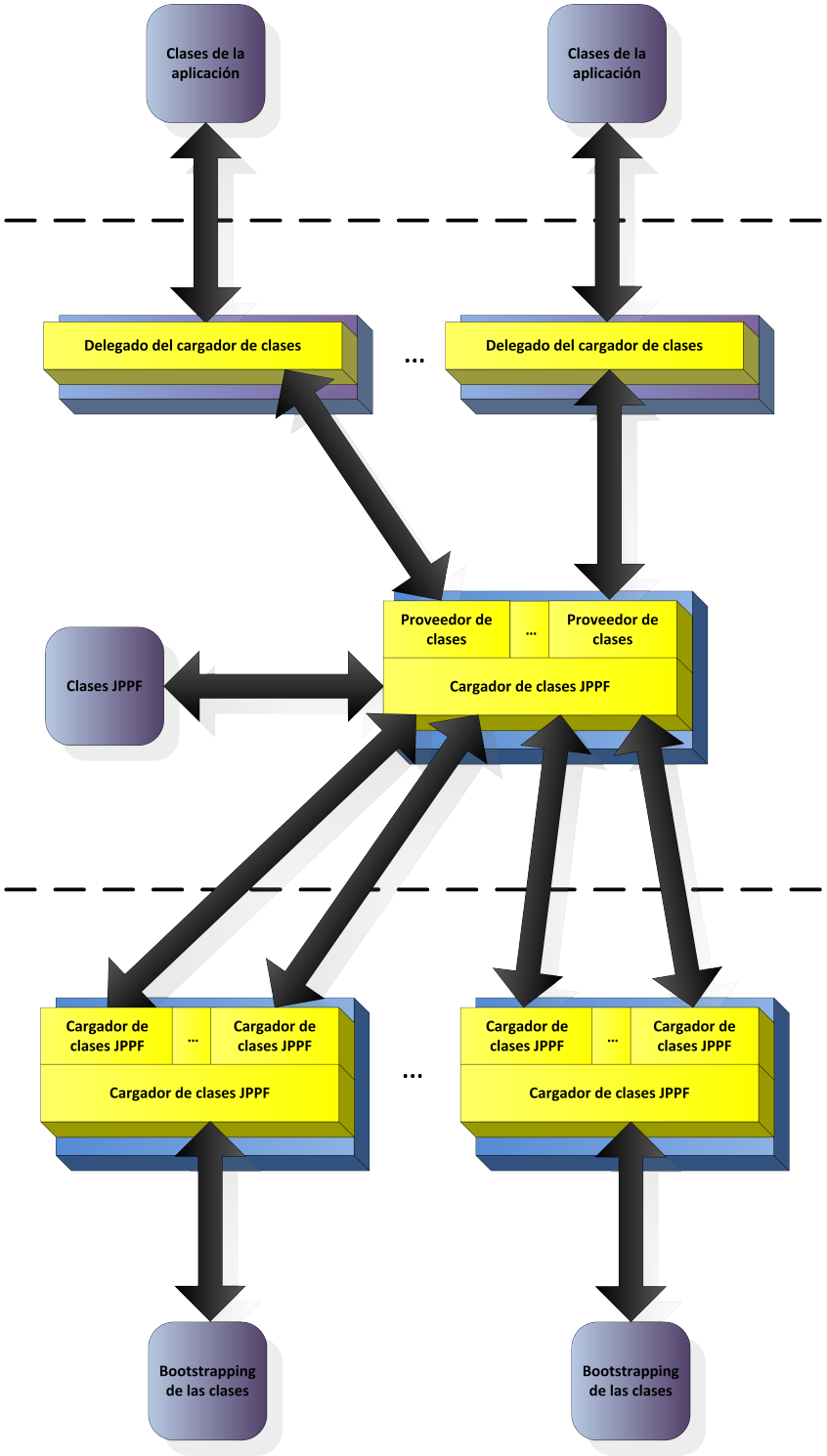


Figura 4.3: Despliegue y carga de clases.

- **Extendiendo la topología del clúster**

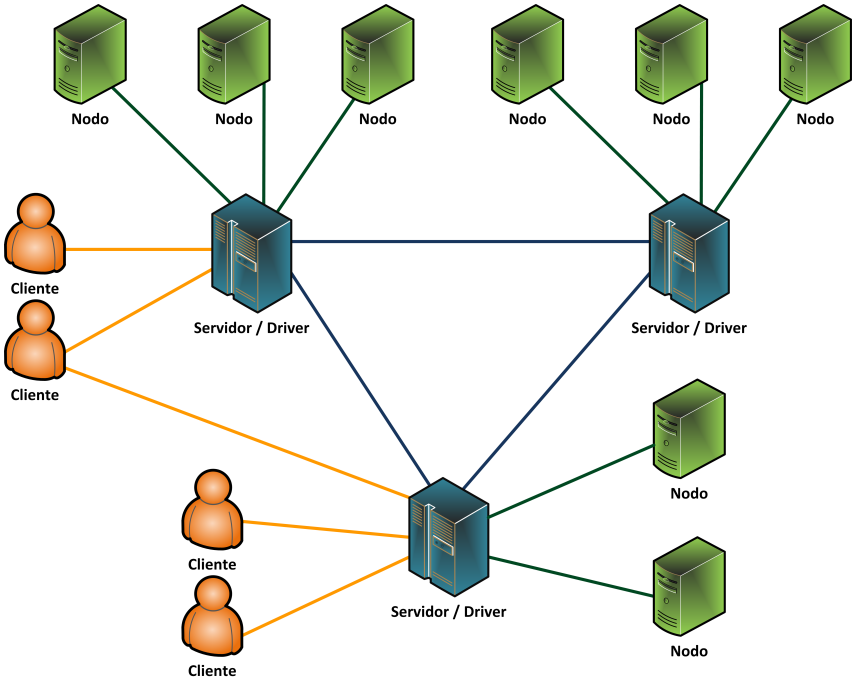


Figura 4.4: Extensión de la topología JPPF.

JPPF puede ser extendido para incluir muchos servidores y comunicarse entre ellos en una topología de punto a punto.

En esta topología, cada servidor es visto por sus compañeros como un nodo, y ve a sus compañeros como clientes.

Hay un gran número de ventajas en este diseño:

- Permite una mayor escalabilidad de la red JPPF, permitiendo complementarse con los servidores adicionales dinámicamente. De esta manera, un servidor puede delegar una parte de su carga a otros servidores.
- No importa cuántos servidores están presentes, los nodos y los clientes se comunican con ellos exactamente de la misma manera.
- La comunicación entre servidores beneficia a las características de conmutación por error y recuperación disponibles en los nodos y los clientes.

4.1.3. Hazelcast

Hazelcast es una plataforma de distribución de datos altamente escalable para Java. Hazelcast permite compartir y crear particiones de los datos de la aplicación fácilmente a través del clúster. Hazelcast es una solución punto a punto (no existe nodo maestro, cada nodo es un

punto) lo que significa que no hay un único punto de fallo. Lo que facilita en gran mayoría el desarrollo y diseño de los grandes sistemas.

Hazelcast es simple. La máquina virtual de Java desde donde se lanzará Hazelcast se convertirá en un clúster dinámico. Aunque de forma predeterminada Hazelcast utiliza *multicast* para el descubrimiento de sus otros nodos, también puede ser configurado para usar únicamente el protocolo *TCP/IP*[1] en entornos donde el *multicast* o no esté disponible o no se quiera llegar a hacer un uso de él.

La comunicación utilizada por Hazelcast entre los miembros de clúster es *TCP/IP* con *Java NIO* (*Java New In/Out*). La configuración por defecto que viene con Hazelcast realiza una copia de seguridad de modo que si un nodo falla, no haya datos que se puedan perder. Es tan sencillo como utilizar la clase de java *java.util* (*Cola, Set, List, Map*). Sólo tiene que añadir la librería *hazelcast.jar* en la ruta de clases y comenzar la codificación.

4.1.3.1. Caracterisiticas

- Características generales:
  - *java.util* (*Cola, Set, List, Map*) distribuido.
  - *java.util.concurrency.locks.Lock distribuido.*
  - *java.util.concurrent.ExecutorService* distribuido.
  - *MultiMap* distribuido.
  - Temas distribuidos de *publicación / suscripción* de mensajería.
  - Soporte para *indexación* y consultas distribuidas.
  - Soporte para *transacciones* e integración de contenedores J2EE (*Java to Enterprise Edition*) vía JCA (*Java EE Connector Architecture*).
  - Encriptación a nivel de *socket* para clústeres seguros.
  - Escrituras y lecturas persistentes para los mapas.
  - Cliente de Java para poder acceder al clúster de forma remota.
  - Sesiones *HTTP* dinámicas.
  - Soporte de la información y eventos de cada uno de los miembros del clúster.
  - Descubrimiento de los miembros de forma dinámica.
  - Escalabilidad dinámica.
  - Copias de seguridad particionadas dinámicamente.
  - Conmutaciones por error de forma dinámica.

[1] Véase más información sobre TCP/IP en la sección 3.4

- Herramientas de seguimiento y monitorización del clúster basadas en web.
- Uso muy simple: incluir un único *jar* al proyecto.
- Súper ligero: menos de un megabyte de tamaño.

- Características comunes a todas las estructuras de datos:

  La distribución de los datos a través de todos los nodos en el clúster es casi uniforme. Por lo que cada nodo realiza x trabajos.

  $x = (1/n \times$ total de datos$) +$ copias de seguridad / n es el número de nodos en el clúster $\forall n \geq 0$, $n \in \mathbb{N}$.

  Si un miembro de la red se cae, su réplica de respaldo que también contiene los mismos datos que los del nodo, será redistribuida dinámicamente consiguiendo como resultado que ningún dato sea perdido.

  Cuando un nuevo nodo entra a participar en el clúster, el nuevo nodo toma la responsabilidad y parte de la carga de los datos que hay en la red. Finalmente el nuevo nodo llevará casi el $(1/n \times$ total de datos$) +$ copias de seguridad, reduciendo así la carga de los demás miembros del clúster.

  No hay un único maestro en el clúster o algo que puede causar un único punto de fallo. Cada nodo del clúster tiene los mismos derechos y responsabilidades. Nadie es superior. No comparte el concepto *maestro/esclavo* por lo que no tiene una dependencia sobre uno o varios servidores.

4.1.3.2. Uso de hazelcast

Hazelcast ayuda a los desarrolladores cuando necesiten:

- Compartir datos o estados entre los muchos servidores, como por ejemplo, la sesión de una aplicación web.
- Una *caché* distribuida para los datos.
- Generar una aplicación clúster.
- Proporcionar una comunicación segura entre los servidores.
- Realizar particiones de los datos en memoria.
- Distribuir la carga de trabajo en muchos servidores.
- Adquirir la ventaja del procesamiento paralelo.
- Ofrecer una gestión de datos a prueba de fallos.

4.1.4. Jasypt

Jasypt es una biblioteca Java que permite a los desarrolladores agregar capacidades básicas de cifrado en sus proyectos con el mínimo esfuerzo, y sin necesidad de tener un conocimiento profundo de cómo funciona la criptografía.

4.1.4.1. Características

- Jasypt sigue los estándares de la criptografía *RSA*[2] basada en contraseñas, y además proporcionada técnicas de cifrado unidireccionales y bidireccionales.
- API abierta para su uso con cualquier proveedor *JCE*, y no sólo para la opción por defecto de la máquina virtual de java. Jasypt puede ser utilizado fácilmente con proveedores tan conocidos como *Bouncy Castle*.
- Seguridad elevada en las contraseñas de los usuarios.
- Compatibilidad con el cifrado binario. Jasypt permite el resumen (*digiest*) y el cifrado de archivos binarios (*matrices de bytes*). A demás, posibilita el cifrado tanto de objetos cómo de archivos cuando sea requerido; cómo por ejemplo, cuando se envían archivos a través de la red.
- Completamente *thread-safe*.
- Proporciona la posibilidad de realizar encriptaciones sin necesidad de realizar configuraciones, así como, herramientas estándar altamente configurables para usuarios avanzados.
- Integración opcional con *Hibernate III* para el cifrado de los campos de una manera encriptada. El cifrado de los campos se define en los archivos de mapeo de *Hibernate*, y sigue siendo transparente para el resto de la aplicación (útil para los datos personales sensibles, bases de datos con muchos usuarios habilitados para leer...). Cifrado de textos, archivos binarios, números, booleanos, fechas...
- Integración perfecta con aplicaciones *Spring*. Todos los resúmenes (*digesters*) y las encriptaciones en Jasypt han sido diseñadas para ser usadas fácilmente desde *Spring*. Y a causa de que Jasypt es *thread-safe*, se puede usar sin tener preocupaciones de sincronización en entornos orientados al *singleton* como *Spring*.
- Integración opcional con *Spring Security*(anteriormente llamado, *Acegui Security*) para una mejor y más segura encriptación de contraseñas a través de un grado mayor de configuración y control.

[2]Más información sobre RSA en la sección 3.3.4.3

- Proporciona un fácil uso de las herramientas CLI(*Command Line Interface*) que permite a los desarrolladores inicializar sus herramientas para el cifrado de los datos, incluyendo operaciones de encriptación, decriptación y resumen de las operaciones de mantenimiento de sus tareas o *scripts*.

- Se integra en *Apache Wicket*, para una mayor robustez en la encriptación de *URL* de aplicaciones seguras.

- Incorpora guías sencillas y documentación en *javadoc*, que permite a los desarrolladores a entender realmente que están haciendo con sus datos.

- Soporte robusto para el juego de caracteres, diseñado para cifrar y resumir adecuadamente los textos, sin llegar a importar el juego de caracteres original que se está usando.

- Soporte completo para idiomas como el japonés, coreano, árabe, sin necesidad de recodificación o problemas en la plataforma.

- Nivel elevado en las características de configuración: El desarrollador puede llegar a implementar trucos como instruir a un encriptador para preguntar, por ejemplo, a un servidor *HTTPS* remoto cual es la contraseña a utilizar para la encriptación. Permitiendo satisfacer todas las necesidades de seguridad.

4.1.5. SQL

El **lenguaje de consulta estructurado** o SQL (por sus siglas en inglés *structured query language*) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas con el fin de recuperar, de una forma sencilla, información de interés de una base de datos, así como también hacer cambios sobre ella. Es un lenguaje de cuarta generación (4GL).

4.1.5.1. Historia

Los orígenes del SQL están ligados a los de las bases de datos relacionales. En 1970 *E. F. Codd* propone el modelo relacional y asociado a éste un sublenguaje de acceso a los datos basado en el cálculo de predicados. Basándose en estas ideas, los laboratorios de *IBM* definen el lenguaje *SEQUEL* (*Structured English QUEry Language*) que más tarde sería ampliamente implementado por el sistema de gestión de bases de datos experimental *System R*, desarrollado en 1977 también por *IBM*. Sin embargo, fue *Oracle* quien lo introdujo por primera vez en 1979 en un programa comercial.

El *SEQUEL* terminaría siendo el predecesor de SQL, siendo éste una versión evolucionada del primero. El SQL pasa a ser el lenguaje por excelencia de los diversos sistemas de gestión de bases de datos relacionales surgidos en los años siguientes y es por fin estandarizado en 1986 por el *ANSI*, dando lugar a la primera versión estándar de este lenguaje, el *SQL-86* o *SQL1*. Al año siguiente este estándar es también adoptado por la *ISO*.

Sin embargo, este primer estándar no cubre todas las necesidades de los desarrolladores e incluye funcionalidades de definición de almacenamiento que se consideraron suprimir. Así que en 1992 se lanza un nuevo estándar ampliado y revisado del SQL llamado *SQL-92* o *SQL2*.

En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los *SGBD* comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar *SQL-92* es general y muy amplio.

4.1.5.2. Características generales

El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones en éstos últimos.

Es un lenguaje declarativo de *alto nivel* o de *no procedimiento*, que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación y la orientación a objetos. De esta forma una sola sentencia puede equivaler a uno o más programas que se utilizarían en un lenguaje de bajo nivel orientado a registros.

- **Optimización:**

  SQL es un lenguaje declarativo. O sea, que especifica qué es lo que se quiere y no cómo conseguirlo, por lo que una sentencia no establece explícitamente un orden de ejecución.

  El orden de ejecución interno de una sentencia puede afectar gravemente a la eficiencia del *SGBD*, por lo que se hace necesario que éste lleve a cabo una optimización antes de su ejecución. Muchas veces, el uso de índices acelera una instrucción de consulta, pero ralentiza la actualización de los datos. Dependiendo del uso de la aplicación, se priorizará el acceso indexado o una rápida actualización de la información. La optimización difiere sensiblemente en cada motor de base de datos y depende de muchos factores.

4.1.6. Java Swing

Swing es una biblioteca gráfica para Java. Forma parte de Java de *Sun Microsystems Foundation Classes* (JFC), un API para proporcionar una interfaz gráfica de usuario(GUI) para programas Java.

Swing fue desarrollado para proporcionar un conjunto más complejo de los componentes de interfaz gráfica de usuario que el anterior *Abstract Window Toolkit*. Esta tecnología proporciona un *look and feel* que emula el aspecto de varias plataformas, y también es compatible diferentes *look&feel* permitiendo que las aplicaciones tengan un aspecto relacionado con la plataforma en la que se está utilizando.

4.1.6.1. Historia

Las *Internet Foundation Classes* (IFC) eran una biblioteca gráfica para el lenguaje de programación Java desarrollada originalmente por *Netscape* y que se publicó en 1996.

Desde sus inicios el entorno Java ya contaba con una biblioteca de componentes gráficos conocida como *AWT*. Esta biblioteca estaba concebida como una API estandarizada que permitía utilizar los componentes nativos de cada sistema operativo. Entonces una aplicación Java corriendo en *Microsoft Windows* usaría el botón estándar de *Windows* y una aplicación corriendo en *UNIX* usaría el botón estándar de *Motif*. En la práctica esta tecnología no funcionó:

- Al depender fuertemente de los componentes nativos del sistema operativo el programador *AWT* estaba confinado a un mínimo denominador común entre ellos. Es decir que sólo se disponen en *AWT* de las funcionalidades comunes en todos los sistemas operativos.

- El comportamiento de los controles varía mucho de sistema a sistema y se vuelve muy difícil construir aplicaciones portables. Fue por esto que el eslogan de Java *Escríbalo una vez, ejecútelo en todos lados* fue parodiado como *Escríbalo una vez, pruébelo en todos lados.*

En cambio, los componentes de *IFC* eran mostrados y controlados directamente por código Java independiente de la plataforma. De dichos componentes se dice con frecuencia que son componentes ligeros, dado que no requieren reservar recursos nativos del sistema de ventanas del sistema operativo. Además al estar enteramente desarrollado en Java aumenta su portabilidad asegurando un comportamiento idéntico en diferentes plataformas.

En 1997, *Sun Microsystems* y *Netscape Communications Corporation* anunciaron su intención de combinar *IFC* con otras tecnologías de las *Java Foundation Classes*. Además de los componentes ligeros suministrados originalmente por la *IFC*, Swing introdujo un mecanismo que permitía que el aspecto de cada componente de una aplicación pudiese cambiar sin introducir cambios sustanciales en el código de la aplicación. La introducción de soporte ensamblable para el aspecto permitió a Swing emular la apariencia de los componentes nativos manteniendo las ventajas de la independencia de la plataforma. También contiene un conjunto de herramientas que nos permiten crear una interfaz atractiva para los usuarios.

4.1.6.2. Características generales

La arquitectura Swing presenta una serie de ventajas respecto a su antecedente *AWT*:

- Amplia variedad de componentes: *JButton*, *JSlider*, *JTable*, etc.
- Aspecto modificable (*look and feel*): Se puede personalizar el aspecto de las interfaces o utilizar varios aspectos que existen por defecto
- Arquitectura *Modelo-Vista-Controlador*: Esta arquitectura da lugar a todo un enfoque de desarrollo muy arraigado en los entornos gráficos de usuario realizados con técnicas orientadas a objetos. Cada componente tiene asociado una clase de modelo de datos y una interfaz que utiliza. Se puede crear un modelo de datos personalizado para cada componente, con sólo heredar de la clase *Model*.
- Objetos de acción (*Action objects*): Estos objetos cuando están activados (*enabled*) controlan las acciones de varios objetos componentes de la interfaz.
- Contenedores anidados: Cualquier componente puede estar anidado en otro. Por ejemplo, un gráfico se puede anidar en una lista.
- Escritorios virtuales: Se pueden crear escritorios virtuales o *interfaz de múltiples documentos* mediante las clases *JDesktopPane* y *JInternalFrame*.
- Bordes complejos: Los componentes pueden presentar nuevos tipos de bordes. Además el usuario puede crear tipos de bordes personalizados.
- Diálogos personalizados: Se pueden crear multitud de formas de mensajes y opciones de diálogo con el usuario, mediante la clase *JOptionPane*.
- Clases para diálogos habituales: Se puede utilizar *JFileChooser* para elegir un fichero, y *JColorChooser* para elegir un color.
- Componentes para tablas y árboles de datos: Mediante las clases *JTable* y *JTree*.
- Potentes manipuladores de texto: Además de campos y áreas de texto, se presentan campos de sintaxis oculta *JPassword*, y texto con múltiples fuentes *JTextPane*. Además hay paquetes para utilizar ficheros en formato *HTML* o *RTF*.
- Soporte a la accesibilidad: Se facilita la generación de interfaces que ayuden a la accesibilidad de discapacitados, por ejemplo en *Braille*.

4.1.6.3. Arquitectura

Swing basa sus componentes en la arquitectura MVC *(Modelo - Vista - Controlador)*. En la figura 4.5 se muestra un diagrama sencillo que describe la relación entre el modelo, la vista y el controlador; donde las líneas sólidas indican una asociación directa, y las punteadas una indirecta.

Este modelo se corresponde con el estado del componente; por ejemplo, en una lista el modelo está definido por todos los elementos de la misma; en cambio, en una caja de texto el modelo está definido por el documento del texto. La vista se refiere a cualquier perspectiva de ese modelo; por ejemplo, en una lista la vista es la presentación gráfica de la misma, y en una caja de texto, ídem. Y el controlador (*manejador de eventos*) es el responsable de la actualización del modelo.

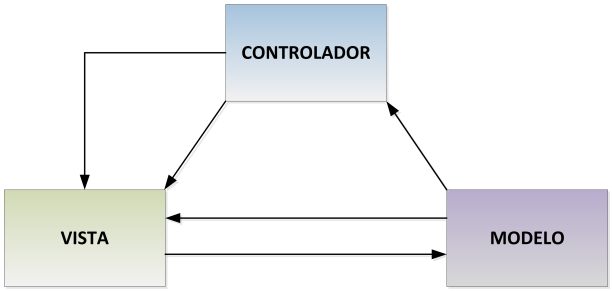


Figura 4.5: Modelo - Vista - Controlador.

Tanto la vista como el controlador son combinados en un objeto denominado *delegado*. Es decir, el delgado representa gráficamente el modelo (trabajo que hace la vista) y transfiere la entrada del usuario hacia éste (trabajo que hace el controlador). De esta forma se evita conocer aspectos internos muy complejos, como la comunicación entre el controlador y la vista.

Un *JComoponet* puede tener diferentes modelos y delegados. Para acceder a los modelos, se dispone de los métodos *setModel*y *getModel*, y *setUI* y *getUI* para acceder a los delegados.

Muchos de los componentes Swing no contendedores tienen modelos; por ejemplo, un botón (*JButton*) tiene un modelo (*ButtonModel*) que almacena el estado del botón (la tecla para acceso rápido, si está pulsado, etc.). Otros componentes tiene múltiples modelos; por ejemplo, una lista (*JList*) utiliza un *ListModel* para almacenar sus elementos y un *ListSelectionModel* para seguir la pista del elemento actualmente seleccionado. No obstante, normalmente no se necesita conocer los modelos que usa un componente; por ejemplo, cuando se utilizan botones normalmente se trata con el objeto *JButton* y no con el objeto *ButtonModel*. La finalidad de esto, es simplemente ofrecer la posibilidad de trabajar más eficientemente con los componentes y para compartir fácilmente datos entre los mismos. Por ejemplo, cuando se trabaja con una lista, puede ser mucho más rápido acceder a los datos actuando directamente sobre el modelo que tener que

esperar a cada petición de datos al modelo. Esto se puede realizar utilizando el modelo por defecto del componente o implementado uno propio.

4.1.7. Apache Frameworks

Apache Fraemworks se compone de librerías como commons-lang, commons-pool, log4j, y muchas más. A continuación se describen las librerías utilizadas en la aplicación DJ HACK.

4.1.7.1. Apache commons proper

Apache commons proper es una de las tres partes (*The Commons Proper*, *The Commons Sandbox*, *The Commons Dormant*) de la que está compuesto el proyecto *Apache Commons*.

El proyecto Apache commons proper se dedica a un objetivo principal: la creación y el mantenimiento de componentes Java reutilizables por ello ha desarrollado una amplia lista de componentes, de los cuales estos son los utilizados en este proyecto:

- Commons Lang:Las bibliotecas estándar de Java no proporcionan métodos suficientes para la manipulación de las clases de su núcleo. *Apache Commons Lang* proporciona estos métodos adicionales.

  Lang brinda toda una serie de utilidades de ayuda para la API de *java.lang*, en particular los métodos de manipulación de cadenas, métodos numéricos básicos, concurrencia, serialización y propiedades del sistema. Adicionalmente contiene mejoras básicas para *java.util.Date* y una serie de servicios dedicados a ayudar con los métodos de construcción, tales como,*hashCode*, *toString* y *equals*.

- Commons DBCP:*Commons DBCP* (*Database connection pooling services*) proporciona al desarrollador los servicios para realizar cualquier tipo de pool de conexiones[3].

- Commons Pool:*Pool* proporciona una API *Object-pooling*, con tres aspectos principales:

  1. Un objeto pool es interfaz genérica que los clientes y desarrolladores pueden utilizar para proporcionar implementaciones pools fácilmente intercambiables.
  2. Un conjunto de herramientas para la creación de objetos pools modulares.
  3. Varias implementaciones pool de propósito general.

- Commons codec:*Commons Codec* proporciona implementaciones de codificadores y decodificadores comunes, como *Base64*, *Hex*, *Phonetic* y *URLs*.

[3]Para más información sobre los pool de conexiones a la base de datos Véase la sección de *agrupamiento de conexiones* 3.5.

- Commons Logging:*Apache Commons Logging* (JCL) nos proporciona una interfaz cuyo propósito es ser una *abstracción* independiente de otros frameworks o herramientas de logeo.

  El propósito es usar esta interfaz en el código y después conectar una implementación en específico (*Log4J*,*Avalon LogKit* , *JDK 1.4 Logging API*) a través de configuración sin necesidad de modificar el código.

4.1.7.2. Log4j

Log4j es una biblioteca *open source* desarrollada en Java por la *Apache Software Foundation* que permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o *logs* a tiempo de ejecución y no a tiempo de compilación como es comúnmente realizado. La configuración de salida y granularidad de los mensajes es realizada a tiempo de ejecución mediante el uso de archivos de configuración externos.

4.1.8. LaTeX

LaTeX es un sistema de composición de textos que está formado mayoritariamente por órdenes (*macros*) construidas a partir de comandos de *TeX* un lenguaje de bajo nivel, en el sentido de que sus acciones últimas son muy elementales pero con la ventaja añadida, en palabras de *Lamport*, de "*poder aumentar las capacidades de LaTeX utilizando comandos propios del TeX descritos en The TeXbook*". Esto es lo que convierte a LaTeX en una herramienta práctica y útil pues, a su facilidad de uso, se une toda la potencia de *TeX*. Estas características hicieron que LaTeX se extendiese rápidamente entre un amplio sector científico y técnico, hasta el punto de convertirse en uso obligado en comunicaciones y congresos, y requerido por determinadas revistas a la hora de entregar artículos académicos.

Su código abierto permitió que muchos usuarios realizasen nuevas utilidades que extendiesen sus capacidades con objetivos muy variados, a veces ajenos a la intención con la que fue creado: aparecieron diferentes dialectos de LaTeX que, a veces, eran incompatibles entre sí. Para atajar este problema, en 1989 *Lamport* y otros desarrolladores iniciaron el llamado *Proyecto LaTeX3*. En otoño de 1993 se anunció una reestandarización completa de LaTeX, mediante una nueva versión que incluía la mayor parte de estas extensiones adicionales (como la opción para escribir transparencias o la simbología de la *American Mathematical Society*) con el objetivo de dar uniformidad al conjunto y evitar la fragmentación entre versiones incompatibles de *LaTeX 2.09*. Esta tarea la realizaron *Frank Mittlebach*, *Johannes Braams*, *Chris Rowley* y *Sebastian Rahtz* junto al propio *Leslie Lamport*. Hasta alcanzar el objetivo final del *Proyecto 3*, a las distintas versiones se las viene denominando *LaTeX2e* (o sea, *versión 2* y un poco

más..."). Actualmente cada año se ofrece una nueva versión, aunque las diferencias entre una y otra suelen ser muy pequeñas y siempre bien documentadas.

Con todo, además de todas las nuevas extensiones, la característica más relevante de este esfuerzo de reestandarización fue la arquitectura modular: se estableció un núcleo central (el compilador) que mantiene las funcionalidades de la versión anterior pero permite incrementar su potencia y versatilidad por medio de diferentes paquetes que sólo se cargan si son necesarios. De ese modo, LaTeX dispone ahora de innumerables paquetes para todo tipo de objetivos, muchos dentro de la distribución oficial, y otros realizados por terceros, en algunos casos para usos especializados.

4.1.8.1. MiKTeX

MiKTeX es una distribución TeX/LaTeX para *Microsoft Windows* que fue desarrollada por *Christian Schenk*.

Las características más apreciables de MiKTeX son su habilidad de actualizarse por sí mismo descargando nuevas versiones de componentes y paquetes instalados previamente, y su fácil proceso de instalación.

4.1.8.2. Características

- Es libre y fácil de instalar.
- Incluye más de 800 paquetes con *fonts*, *macros*, etc.
- Tiene un visor propio de archivos *dvi* denominado *Yap*.
- Su código es abierto.
- Posee compiladores TeX y LaTeX, convertidores para generar archivos *postscripts*, *pdf* , *html*, etc.; y herramientas para generar bibliografías e índices.
- Posee tres formas de instalación: reducida y completa.

4.2. Herramientas

4.2.1. NetBeans 6.9.1

NetBeans se refiere a una plataforma para el desarrollo de aplicaciones de escritorio usando Java y a un *entorno de desarrollo integrado* (IDE) desarrollado usando la plataforma NetBeans.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos.
Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (*manifest file*) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. *Sun MicroSystems* fundó el proyecto de código abierto NetBeans en junio 2000 y ahora *Oracle* es el patrocinador principal de los proyectos.

Se ha determinado usar este entorno de desarrollo, ya que NetBeans aporta las herramientas necesarias para la realización del proyecto, así como una interfaz sencilla y amigable que permite al desarrollador realizar una implementación organizada y cómoda. Además NetBeans es una plataforma de código abierto lo que supone reducir el coste del proyecto.

4.2.2. MySQL Sever 5.1

MySQL 5.1 es una base de datos relacional, multihilo y multiusuario que se ha estandarizado al ser el soporte de la gran mayoría de *CMS* junto con el conocido lenguaje *PHP*. Algunos de los sistemas que utilizan MySQL 5.1 son *Drupal*, *phpBB* o *Wordpress*, gracias a su dinamismo, versatilidad y a la capacidad de adaptación y mejora que ha demostrado versión tras versión. *MySQL AB* desde enero de 2008 una subsidiaria de *Sun Microsystems* y ésta a su vez de *Oracle Corporation* desde abril de 2009 desarrolla MySQL como software libre en un esquema de licenciamiento dual.

Como es de esperar, esta base de datos soporta todo tipo de instrucciones y operaciones SQL como *búsquedas*, *modificaciones* y *triggers*. Además, ofrece compatibilidad completa con *Unicode* y se adapta a las reglas *ACID* en motores como *InnoDB*, *BDB* y *Clúster*.

Por un lado se ofrece bajo la *GNU GPL* para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en *ANSI C*.

Al contrario de proyectos como *Apache*, donde el software es desarrollado por una comunidad pública y el *copyright* del código está en poder del autor individual, MySQL es patrocinado por una empresa privada, que posee el *copyright* de la mayor parte del código.

Esto es lo que posibilita el esquema de licenciamiento anteriormente mencionado. Además de la venta de licencias privativas, la compañía ofrece soporte y servicios. Para sus operaciones

contratan trabajadores alrededor del mundo que colaboran vía Internet. *MySQL AB* fue fundado por *David Axmark*, *Allan Larsson* y *Michael Widenius*.

MySQL ha sido elegido como gestor de la base de datos para la aplicación por ser una herramienta de código abierto fácil de usar, que además de abaratar el presupuesto, proporciona las conexiones rápidas y seguras que requiere DJ HACK para cumplir sus objetivos.

4.2.3. IP Sniffer

IP Sniffer es una de las herramientas de la suite de *IP Tools*, encargada de analizar protocoles de red.

IP tools incorpora en su suite además de IP Sniffer muchas otras herramientas: Monitor de trafico IP, estadísticas IP, *ARP*(listas y eliminación de entradas, envío de solicitudes), impresión de rutas, *spoofing*(TCP, UDP, ICMP, ARP), resolución de direcciones IP, PING, escaneo TCP, etc.

Este programa ha sido elegido por ser una herramienta gratuita utilizada para realizar auditorías de red; como controlar el envío de tráfico entre los componentes del clúster y realizar capturas de paquetes con la finalidad de comprobar el correcto funcionamiento de la encriptación de las transmisiones.

4.2.4. Apache Ant

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.

Esta herramienta, hecha 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.

4.2.5. TeXnicCenter

TeXnicCenter es un editor software libre de LaTeX para *Windows*, el cual integra en sí mismo las herramientas necesarias para la composición de texto científico, desde una ventana de compilación integrada, una ayuda completa y manual de LaTeX para los usuarios primerizos, así como un entorno personalizable para los usuarios avanzados.

TexnicCenter está diseñado para trabajar con la distribución *MiKTeX*. Por ello, tras instalarse, TeXnicCenter reconoce los ejecutables de *MiKTeX* y establece las rutas a los compiladores de consola de esta distribución.

De forma similar, en caso de estar disponible también se puede configurar un visor para los documentos *PDF* generados, como *Adobe Acrobat Reader/Professional* o *Foxit Reader*. *Acrobat* puede configurarse para cerrar un documento *PDF* al recompilarlo con LaTeX. Estas operaciones pueden llevarse a cabo mediante llamadas *Dynamic Data Exchange*.

Se ha determinado el uso de este entorno de desarrollo para realizar la documentación por ser básicamente gratuito e incorporar herramientas que ayudan a maquetar un documento profesional de manera sencilla.

4.2.6. Ghostscript 8.71

Ghostscript, escrito por *Peter Deutsch*, fundador de *Aladdin Enterprises*, es el programa intérprete por excelencia de documentos en formato *PS* (y también *PDF*).

Ghostscript permite presentar datos *PS* y *PDF* en la pantalla y además traducirlos de manera que puedan ser impresos en una impresora con capacidad gráfica mediante el uso del controlador de dicha impresora.

Aladdin Enterprises mantiene las nuevas versiones de Ghostscript hasta que alcanzan una cierta edad, y luego las libera a la *Fundación de Software Libre* (*FSF, Free Software Foundation*) para que se distribuya como *GNU Ghostscript*. Estas versiones son las que se distribuyen "libres de cargo.[en] las distribuciones de Linux.

Dispone de una serie de dispositivos controladores para diversos tipos de impresora. Si invocamos *gs -h*, nos aparecerá gran cantidad de información sobre nuestra versión instalada de Ghostscript, entre la que encontraremos los dispositivos controladores.

Esta herramienta ha sido utilizada por su uso extendido y por su fácil integración con cualquier editor Latex.

4.2.7. GWView 4.9

GSview es un interfaz gráfico para utilizar Ghostscript en *MS-Windows* o *OS/2*.

Para los documentos construidos siguiendo las especificaciones de *Adobe* acerca de la estructura de los documentos *PostScript* (*DSC*), GSview permite seleccionar las páginas que serán visualizadas o impresas.

4.2.7.1. Características

- Muestra e imprime archivos *PostScript* y *PDF*.
- Permite ver páginas en un orden arbitrario (Próxima, Previa, Ir a)..
- El tamaño de la página y la orientación son automáticamente seleccionadas desde los comentarios *DSC* o utilizando el menú.
- Imprime páginas seleccionadas usando *Ghostscript*.
- Convierte páginas a *bitmap*, *PDF* o *PostScript*.
- Botón de zoom.
- Extrae páginas seleccionadas a otro archivo.
- Copia imágenes *Bitmap* del portapapeles como archivo *BMP*.
- Extrae texto o busca por texto.

4.2.8. Gimp 2.6

GNU Image Manipulation Program(GIMP) es un programa de edición de imágenes digitales en forma de mapa de bits, tanto dibujos como fotografías. Es un programa libre y gratuito. Está englobado en el proyecto *GNU* y disponible bajo la *Licencia pública general de GNU*.

Es el programa de manipulación de gráficos disponible en más sistemas operativos, como ser, *Unix*, *GNU/Linux*, *Windows*, *Mac OS X*, entre otros, además se incluye en muchas distribuciones *GNU/Linux*.

La interfaz de GIMP está disponible en varios idiomas, entre ellos: Español, alemán, inglés, catalán, gallego, euskera, francés, italiano, ruso, sueco, noruego, coreano, neerlandés y en otras lenguas adicionales.

Gimp es una alternativa firme, potente y rápida a *Photoshop* para muchos usos, aunque no se ha desarrollado como un clon de él y posee una interfaz bastante diferente.

Este programa ha sido adquirido y utilizado para la realización del desarrollo y diseño del logotipo de DJ HACK, así como el tratamiento gráfico de otros muchos componentes de la aplicación. Al ser totalmente gratuito, abarata el presupuesto del proyecto de manera considerable.

4.2.9. Microsoft Visio 2010

Microsoft Visio es un software de dibujo vectorial para *Microsoft Windows*. Visio comenzó a formar parte de los productos de *Microsoft* cuando fue adquirida la compañía *Visio* en el año 2000.

Las herramientas que lo componen permiten realizar diagramas de oficinas, diagramas de bases de datos, diagramas de flujo de programas, UML, y más, que permiten iniciar al usuario en los lenguajes de programación.

Aunque originalmente apuntaba a ser una aplicación para dibujo técnico para el campo de ingeniería y arquitectura; con añadidos para desarrollar diagramas de negocios, su adquisición por *Microsoft* implicó drásticos cambios de directrices de tal forma que a partir de la versión de Visio para *Microsoft Office 2003* el desarrollo de diagramas para negocios pasó de añadido a ser el núcleo central de negocio, minimizando las funciones para desarrollo de planos de ingeniería y arquitectura que se habían mantenido como principales hasta antes de la compra. Una prueba de ello es la desaparición de la función *property line* tan útil para trabajos de agrimensura y localización de puntos por radiación, así como el suprimir la característica de ghost shape que facilitaba la ubicación de los objetos en dibujos técnicos. Al parecer *Microsoft* decidió que el futuro del programa residía en el mundo corporativo de los negocios y no en las mesas de dibujo de arquitectos e ingenieros compitiendo con productos como *AutoCad*, *DesignCad*, *Microstation*, etc.

Microsoft Visio ha sido utilizada para la realización de esquemas y gráficos por ser muy intuitiva y permitir la creación de todo tipo de diagramas de manera fácil y rápida.

4.2.10. Microsoft Project 2010

Microsoft Project (o MSP) es un software de administración de proyectos diseñado, desarrollado y comercializado por *Microsoft* para asistir a administradores de proyectos en el desarrollo de planes, asignación de recursos a tareas, dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo.

El software *Microsoft Office Project* en todas sus versiones (la versión 2010 es la más reciente) es útil para la gestión de proyectos, aplicando procedimientos descritos en el PMBoK (*Management Body of Knowledge*) del PMI (*Project Management Institute*).

Esta herramienta ha sido utilizada por su uso extendido, lo que facilita la formación, además de permitir llevar a cabo gran parte del proceso de gestión del proyecto.

Parte III

La aplicación DJ HACK: diseño, estructura y gestión

CAPÍTULO

5 Diseño y estructura

Este capítulo describe los principales objetivos de DJ HACK, los métodos utilizados para alcanzar dichos objetivos, la arquitectura empleada, problemas a los se ha tenido que enfrentar el desarrollador y sus posibles soluciones.

5.1. Objetivo 1: Diseño e implementación de un algoritmo de fuerza bruta

La principal característica de DJ HACK es la posibilidad de realizar ataques a claves hash criptográficas a través de entornos distribuidos. Para ello, se ha diseñado y realizado un algoritmo divisible y eficiente es capaz de realizar este tipo de ataques en el intervalo de tiempo más corto posible teniendo en cuenta en entorno que se va a ejecutar.

5.1.1. Descripción general

Antes de profundizar con el área técnica es imprescindible conocer la metodología utiliza por DJ HACK para la realización de ataques de fuerza bruta.

El pilar de esta aplicación se basa en la generación continua de *variaciones con repetición*[1] de *palabras* de una manera organizada. A lo que se denomina *palabra* en DJ HACK es la estructura con un determinado número de posiciones (*longitud de palabra*) donde cada una de ellas representa un elemento del alfabeto a utilizar en el ataque.

Una palabra tiene dos características fundamentales:

Longitud: Representa el tamaño de la palabra. Es decir, cuando se realiza un ataque de fuerza bruta, se debe de indicar cuantas posiciones puede llegar a tener la clave decriptada.

Alfabeto: Estructura representativa del conjunto de caracteres y números se utilizan para generar una palabra. Por ello, cada posición de cada palabra contiene un único elemento del alfabeto.

Sea $a_1a_2 \dots a_i \dots a_n$ la estructura de la palabra / i representa la posición del elemento del diccionario $\forall i \in \mathbb{N}$, n es la longitud de la palabra $\wedge n \in \mathbb{N}$.
Sea $b_1b_2 \dots b_j \dots b_s$: la estructura del diccionario / $\forall j \in \mathbb{N}$

La idea es que según se van generando las palabras, se codifiquen con la encriptación hash deseada por el usuario MD5,SHA1 ... y se vayan comparando con la clave cifrada hasta encontrar una coincidencia. En el caso de no haberla encontrado el algoritmo habrá procesado b^n palabras $/b$ es el tamaño de la palabra $\forall b \in \mathbb{N}$ y n es la longitud del alfabeto $\forall n \in \mathbb{N}$, ya que esta técnica se encarga en la generación de variaciones con repetición.

5.1.2. Recursividad vs Iteración

La necesidad de crear todas las variaciones posibles asignando cada uno de los elementos del alfabeto por cada una de las posiciones de la estructura de la palabra, ha requerido un estudio previo de mitologías a emplear en el desarrollo del algoritmo.

Durante la realización del proyecto se han estudiado dos metodologías posibles para realizar la generación de palabras que se utiliza en el ataque de fuerza bruta.

A continuación se describen las técnicas empleadas en su implementación, así como, las ventajas e inconvenientes de dichas técnicas.

[1] Véase la seccion variaciones con repetición 3.6.1

5.1.2.1. Recursividad

La recursividad es la forma en la cual se especifica un proceso basado en su propia definición. Siendo un poco más precisos, y para evitar el aparente círculo sin fin en esta definición:

Un problema que pueda ser definido en función de su tamaño, sea este N, pueda ser dividido en instancias más pequeñas $(< N)$ del mismo problema y se conozca la solución explícita a las instancias más simples, lo que se conoce como casos base, se puede aplicar inducción sobre las llamadas más pequeñas y suponer que estas quedan resueltas.

5.1.2.2. Iteración:

En programación, la *iteración* es la repetición de una serie de instrucciones en un programa de computadora. Puede usarse tanto como un término genérico (como sinónimo de repetición) así como para describir una forma específica de repetición con un estado.

5.1.2.3. Ventajas e inconvenientes

En esta parte se comparan los dos enfoques y se examinan las razones por las que el programador puede elegir u enfoque u otro según la situación específica.

Tanto la *iteración* como la *recursión* se basan en una estructura de control: la iteración utiliza una estructura repetitiva y la recursión utiliza una estructura de selección. La iteración y la recursión implican ambas repetición: la iteración utiliza explícitamente una estructura repetitiva, mientras que la recursión consume la repetición mediante llamadas repetidas. La iteración y la recursión implican cada una un test mientras que la recursión termina cuando se reconoce un caso base o la condición de salida se alcanza.

La recursión tiene muchas desventajas. Se invoca repetidamente al mecanismo de recursividad y en consecuencia se necesita tiempo suplementario para realizar las mencionadas llamadas.

Esta característica puede resultar cara en tiempo de procesador y espacio de memoria. Cada llamada de una función recursiva produce otra copia de la función (realmente solo las variables de función) sea creada; esto puede consumir memoria considerable. Por el contrario, la iteración se produce dentro de una función, de modo que las operaciones suplementarias de las llamadas a la función y asignación de memoria adicional son omitidas.

En consecuencia, existen numerosos problemas complejos que poseen naturaleza recursiva y, a su vez, son más fáciles de implementar con algoritmos de este tipo. Sin embargo, en condiciones críticas de tiempo y de memoria, es decir, cuando el consumo de tiempo y memoria sean decisivos o concluyentes para la resolución del problema, la solución a elegir debe ser, normalmente la iterativa.

5.1.2.4. Metodología empleada

De por sí, cualquier algoritmo de fuerza bruta requiere un alto coste computacional (carga elevada de la CPU, carga elevada de la memoria RAM, etc.), requerido por el número de variaciones de palabras que debe generar el programa para intentar decriptar una clave. Por ello, una vez contempladas las ventajas e inconvenientes de ambas tecnologías se ha optado por la iteración como mecanismo de desarrollo para realizar el algoritmo. Lo que supone realizar un desarrollo más complejo y sofisticado pero eficiente y sin una carga computacional excesiva.

5.1.3. Diseño e implementación

El diseño y la arquitectura del algoritmo es crucial para la aplicación, pues determina la rapidez y efectividad que tendrá DJ HACK a la hora de realizar ataques de fuerza bruta. Por ello se ha diseñado un sistema muy óptimo capaz de generar palabras de manera ordenada y rápida teniendo especial control con la carga elevada que sufren tanto la CPU como la RAM a la hora de realizar este tipo de ataque tan pesado.

5.1.3.1. Funcionamiento inicial

El diseño se basa en *sistemas de numeración posicionales*[2]. La idea para conseguir obtener todas las variaciones de palabras posibles de una determinada longitud se ha obtenido mediante la definición de un sistema de numeración y a los métodos de cálculo de bases.

Para ello se diseña el concepto de la *palabra matemática* donde cada palabra está formada por una *array numérico*. Cada posición del array numérico o palabra matemática hace referencia a una posición de alfabeto. Con lo cual, cada una de las palabras está compuesta por referencias a cada uno de los caracteres del alfabeto. Así pues, todos los elementos del array forman un número en la base establecida por la longitud del alfabeto, representado cada elemento del array una posición del alfabeto.

Sea alfabeto = $b_1 \quad b_2 \quad \dots \quad b_r$ / r s el número de caracteres del alfabeto $\forall r \geq 0 \in \mathbb{N}$ donde cada elemento del alfabeto es distinto.

Sea $c \in r$ la base de trabajo.

Sea $array_i = a_1 \quad a_2 \quad \dots \quad a_n$ / $\forall a \in r$ y n es el número de caracteres de la palabra $\forall n \in \mathbb{N}$, $\vee \forall i \in \mathbb{N} i \leq r^n$

Una vez obtenido el diseño, el realizar todas las variaciones de palabras posibles, el algoritmo se encarga de generar todas las variaciones de palabras matemáticas, empezando desde la primera palabra $0 \quad 0 \quad \dots \quad 0$, a continuación se ira sumando una unidad hasta llegar a la $r \quad r \quad \dots \quad r$ teniendo en cuenta el sistema de numeración empleado, donde la base de trabajo

[2] Véase la sección 3.6.2.1

Sea $b \in r$ la base de trabajo.
Sea y el intervalo en base decimal.
Sea $y_{b_r} = b_1 \quad b_2 \quad \dots \quad b_i \quad \dots \quad b_n$ / $n \in \mathbb{N}$ el intervalo en base r.
Sea x = $a_1 \quad a_2 \quad \dots \quad a_i \quad \dots \quad a_n$ la palabra inicial / $\forall i \in \mathbb{N}$, $0 \leq i \leq r$ y $n \in r$
Sea z el número de palabras auxiliares.
Sea C el número de palaras auxiliares / $1 \leq C \leq r^n$.
Sea $C_i = c_1 \quad c_2 \quad \dots \quad c_k \quad \dots \quad c_n$ una palabra auxiliar $\forall i \in \mathbb{N}$.
Con lo cual la lista de palabras auxiliares se forman de la siguiente manera:
$C_0 = 0 \quad 0 \quad \dots \quad 0$
$C_1 = C_0 + b_1 \quad b_2 \quad \dots \quad b_n$
$C_2 = C_1 + b_1 \quad b_2 \quad \dots \quad b_n$
...
$C_i = C_{i-1} + b_1 \quad b_2 \quad \dots \quad b_n$
...
$C_s = C_{n-1} + b_1 \quad b_2 \quad \dots \quad b_n$ / $\forall s \in \mathbb{N}$ y $0 \leq s \leq z$

es la longitud del alfabeto. Después de todo el cálculo matemático se llegan a tener r^n arrays. Dependiendo de los valores r y n el número de arrays a generar puede llegar a ser bastante elevado, por ello los programas de fuerza bruta no distribuidos son tan lentos, pues el gran número variaciones a generar hacen que tanto la CPU como la RAM sufran sobrecalentamiento debido a la sobrecarga.

5.1.3.2. Divisibilidad

Una vez desarrollado el algoritmo base que generará todas las variaciones de palabras posibles, se ha desarrollado un mecanismo capaz de dividir esas tareas tan pesadas en parte más pequeñas destinadas a procesarse en computadoras distintas. La idea es establecer tres conceptos determinativos:

- Intervalo: Es la cantidad de palabras a generar por cada hilo de cada núcleo del procesador de cada terminal de una manera ordenada.
- Palabras auxiliares: Las palabras auxiliares siguen la misma estructura que las palabras matemáticas. Cada una de ellas representa la palabra matemática inicial por la que se deberán generar las sucesivas variaciones en cada uno de los nodos. Este tipo de palabras no son consecutivas ($0 \quad 0 \quad \dots \quad 0$, $0 \quad 0 \quad \dots \quad 1$, ... , $r \quad r \quad \dots \quad r$) si no que forman teniendo en cuenta el intervalo establecido:
- Número de palabras auxiliares: Es el número de palabras auxiliares que genera la computadora donde se está lanzado el ataque

Con lo cual el funcionamiento es bastante simple, la idea es enviar sucesivamente secuencias de palabras auxiliares generadas por la computadora central, al servidor encargado de distribuirlas por los terminales, hasta haber finalizado todas las variaciones posibles.

Debido a la que la generación de *palabras auxiliares* también puede llegar a resultar una tarea bastante pesada, no se calculan de golpe todas las palabras auxiliares posibles, sino tantas palabras auxiliares como indique el usuario. Cada vez que se obtiene una secuencia de palabras auxiliares son enviadas al servidor que las despliega y distribuye por cada nodo de la red.

Con ello cada hilo de cada núcleo del procesador o procesadores de cada terminal a partir de la palabra auxiliar recogida se encargará de generar de manera ordenada tantas palabras como este definido en el intervalo.

Con lo que a través de la combinación del número de palabras auxiliares y el intervalo, se puede configurar en cierta medida la carga que soportará cada uno de las computadoras a la hora de realizar el ataque.

5.1.3.3. Transformación y hasheo

Tras los conceptos básicos vistos en el apartado anterior. La transformación de las palabras matemáticas a palabras reales es bastante sencilla debido a que cada elemento de cada palabra matemática hace referencia a una posición del alfabeto, con lo cual no se trata más que sustituir dicha referencia por el símbolo que ocupa del alfabeto, convertir ese *array* a *string*, hashearlo con la codificación definida en una base hexadecimal[3], y compararlo con la clave codificada.

En el siguiente método de DJ HACK muestra cómo se realiza la transformación y el hasheo de palabras:

```
        /**
* <p>Dado una pPalabra de enteros, la convierte a su
   correspondiente pPalabra
* en carateres usando el alfabeto elegido.</p>
*
* <p>PRE: Los enteros de cada posición del array no pueden ser
   mayores que la
* longitud del alfabeto, ya que si un numero contendio en dicho
   array
* excediese de la longitud total del alfabeto, no corresponderia
    a ningun
* caracter de dicho alfabeto.</p>
*
* @param pPalabra La pPalabra de enteros.
* @param pAlfabeto El alfabeto.
* @return La palabra compuesta por los caracteres del alfabeto.
*/
```

[3]DJ HACK no soporta otro tipo de bases

```
private String traducirPalabraAString(int[] pPalabra, String
    pAlfabeto) {
     String palabraTraducida = "";
     for (int i = 0; i
             < pPalabra.length; i++) {
         palabraTraducida = palabraTraducida + pAlfabeto.charAt(
            pPalabra[i]);
     }
     return palabraTraducida;
}
```

5.2. Objetivo 2: Implantación del algoritmo en una red distribuida

Una vez desarrollado el algoritmo el siguiente paso a dar es distribuirlo a través de la red. Para ello entra en juego el framework de *JPPF* y *Hazelcast*. La idea es enviar un conjunto de palabras auxiliares al servidor y esté se encarga de distribuirlas a través de los nodos donde son procesadas. Una vez se hayan procesado las palabras, los resultados son devueltos al servidor y posteriormente al cliente. El cliente, comprueba los resultados obtenidos, y si aún no se ha encontrado la clave, vuelve a realizar los pasos anteriores; es decir, computa y envía al servidor el siguiente bloque de palabras auxiliares. Está metodología de trabajo se realiza hasta encontrar la clave o hasta que se hayan procesado todas las palabras auxiliares.

5.2.1. Implantación de JPPF

La implantación de JPPF se basa en tres objetivos fundamentales. En la escritura de la *tarea atómica*, la escritura de un *trabajo*(*job*) donde se introducen las tareas que se quieren procesar paralelamente y el *despliegue* de las tareas en el grid.

5.2.1.1. Escritura una tarea JPPF

Lo primero es desarrollar una tarea JPPF encargada de generar tantas palabras matemáticas como estén definidas por el usuario a partir de la palabra auxiliar enviada, y según se vayan realizando las variaciones ir hasheandolas e ir comparándolas con la clave original hasta encontrar una coincidencia o haber acabado de procesarlas.

Una tarea JPPF es la unidad más pequeña de código que puede ejecutarse en una red JPPF. Desde una perspectiva JPPF, se define como una unidad de código atómico. Una tarea siempre

se define como una subclase de la clase *JPPFTask*. *JPPFTask* es una clase abstracta que implementa la interfaz *Runnable*, con lo que la parte de la tarea que se ejecutará en el grid está escrita en su método *run()*.

Desde el punto de vista del diseño, la escritura de una tarea JPPF consta de dos pasos principales: Crear una subclase de *JPPFTask* y aplicar el método *run()*. A continuación se muestra un ejemplo de una clase *JPPTask* sencilla:

```
public class TareaSencilla extends JPPFTask{

    public void run()
       {
       System.out.println("Hola");
       setResult("La ejecución se ha realizado con éxito");
       }

}
```

5.2.1.2. Escritura de un job

Una vez desarrollada la clase que extiende de *JPPTask* a la hora de realizar el ataque de fuerza bruta la idea es crear tantos objetos extiendan de *JPPFTask* como palabras auxiliares haya, donde cada objeto disponga una única palabra auxiliar.

Obtenida la lista de tareas que se quieren distribuir a través de la red, el siguiente paso a dar es encapsularlas en un *job*(*trabajo*) para su posterior envío al servidor.

Un *job* o *trabajo*, es un conjunto de tareas con características comunes entre sí, como un SLA(*Service Level Agreement*) común a todas las tareas. Estas características incluyen: los datos compartidos entre las tareas; la prioridad; el máximo número de nodos que un *job* puede ejecutar, un indicador de suspensión, que permite la presentación de un trabajo en estado de suspensión, en espera de un comando externo para reanudar o comenzar su ejecución y un indicador de bloqueo encargado de especificar si la ejecución del trabajo es síncrono o asíncrono desde el punto de vista de la aplicación. En el siguiente ejemplo se puede ver la realización de un trabajo. Resumidamente, un *job* es la estructura donde se almacenan todas las tareas a desplegar por el clúster.

```
public JPPFJob createJob() throws Exception
  {
    // Crear el trabajo
    JPPFJob job = new JPPFJob();

    // Añadir al job un identificador
```

```
    job.setId("Tarea sencilla Id");

    // Añadir una tarea al trabajo
    job.addTask(new TareaSencilla());

    // Añadir más tareas si es necesario

    return job;
  }
```

Como se puede ver en el ejemplo la creación de un *job* se basa en llamar al constructor por defecto. La llamada al método *job.setId(Tarea sencilla Id)* es usada para añadir al trabajo un nombre que se pueda usar después para monitorizarlo. Si no se llama a este método, el identificador es generado automáticamente, como una cadena de 32 caracteres hexadecimales.

A través del método *addTask(Object task, Object args)* se añaden las tareas al trabajo.

Aplicando esta metodología en DJ HACK, una vez procesadas las palabras auxiliares, cada una de ellas es almacenada (a través del constructor del objeto que hereda de la clase *JPPTask*) en una tarea. Y a través del método *addTask(Object task, Object args)* son añadidas al trabajo que posteriormente será enviado al servidor.

5.2.1.3. Despliegue de tareas

Ahora que se ha explicado cómo crear tareas y la manera de añadirlas a un *job*, el siguiente paso a realizar es ejecutar ese trabajo en el grid y procesar los resultados de esa ejecución.

A la hora de realizar el ataque de fuerza bruta la idea es crear tantos objetos extiendan de *JPPFTask* como palabras auxiliares haya, donde cada objeto disponga una única palabra auxiliar.

A través del constructor *JPPFClient()* se inicializa el framework de JPPF en la aplicación. Cuando es ejecutado se leen los ficheros de configuración, a continuación se estabiliza la conexión con uno o más servidores para la ejecución del trabajo, así como la monitorización y mantenimiento de la conexión con cada servidor conectado al cliente. Por último se registran oyentes para controlar el estado de la conexión. Con lo cual a través de este método se conecta el *cliente* a los servidores; cliente ubicado en el núcleo del sistema de fuerza bruta.

La inicialización *cliente* tiene un impacto no despreciable en la carga de memoria y en los recursos de la red. Es por eso en esta aplicación se ha decláralo como instancia única para usar siempre el mismo *cliente* en toda la aplicación, asegurando una mayor escalabilidad ya que también está diseñado para su uso simultáneo por parte de varios procesadores.

Teniendo en cuenta los ejemplos anteriores, después de inicializar el *cliente* JPPF, el si-

guiente paso es enviar el trabajo, como se muestra en el siguiente ejemplo:

```
public void desplegarTareas(JPPFJob job) throws Exception
  {
    // Establecer el trabajo en modo bloqueo
    job.setBlocking(true);

    // Submit the job and wait until the results are returned.
    // The results are returned as a list of JPPFTask instances,
    // in the same order as the one in which the tasks where
       initially added the job.

// Envio del trabajo y espera hasta obtener los resultados.
    List<JPPFTask> results = jppfClient.submit(job);

    //Procesar los resultados
    for (JPPFTask task: results)
    {
      // Procesar posibles exceptions de las tareas
      if (task.getException() != null)
      {
              }
      else
      {
        //Procesar resutlados
      }
    }
```

La primera sentencia de este método capacita al trabajo de ejecutarse en modo bloqueo, es decir, el programa queda totalmente congelado hasta que los nodos hayan acabado de realizar todas las tareas.

```
    job.setBlocking(true);
```

La segunda declaración es la que va a enviar el trabajo al servidor y esperar hasta que haya que se devuelvan los resultados:

```
List<JPPFTask> results = jppfClient.submit(job);
```

Se puede ver que los resultados se devuelven como una lista de objetos *JPPFTask*. Aunque no existe ninguna garantía del orden de ejecución de las tareas en el grid, se garantiza que cada

tarea en la lista de resultados va a estar ubicada en la misma posición en la que se ha añadido al trabajo. En otras palabras, los resultados están siempre en el mismo orden que las tareas en el trabajo.

El último paso consiste en interpretar y procesar los resultados. Desde el punto de vista JPPF, hay dos posibles resultados de la ejecución de una tarea: una que plantea una *Throwable*, y uno que no la plantea. Las tareas pueden sufrir errores de manera que no se llegue a un resultado; para tener notificación por parte de la aplicación de que alguna tarea no se ha realizado con éxito es necesario controlarlo a través del método *getException()*.

Los resultados reales de la computación de una tarea pueden ser cualquier atributo de la tarea, o cualquier objeto accesible de ellos. La API *JPPFTask* ofrece dos métodos de conveniencia para ayudar a hacer esto: *setResult(Object)* y *getResult()*, sin embargo no es obligatorio su uso, y se puede aplicar el resultado en cualquier estructura o atributo de la propia clase.

Como ejemplo se ha modificado esta parte del código para mostrar el mensaje de excepción si la excepción es elevada, o mostrar el resultado de otro modo:

```
if (task.getException() != null)
  {
    System.out.println("Ha ocurrido una excepción: "
      + task.getException().getMessage());
  }
  else
  {
    System.out.println("Resultados de la ejecución: "
      + task.getResult());
  }
```

El despliegue de las tareas en DJ HACK es algo más complejo que el mostrado en los ejemplos anteriores debido al constante procesamiento de palabras auxiliares y posterior envío del trabajo.

Como se ha comentado en el apartado anterior, una vez generadas las palabras auxiliares estas son encapsuladas en un *job*. A través del método *submit(JPPFJob)* el trabajo se envía al servidor encargado de distribuir las tareas por cada nodo mientras se espera en modo bloqueado a obtener los resultados. Una vez los nodos hayan acabado de ejecutar las tareas, estas son devueltas al cliente con tres resultados posibles que acarrean tres acciones totalmente distintas:

El primer posible resultado es que se haya producido algún error en la ejecución de una tarea. Cuando se produce esta situación, la tarea que ha generado la excepción se almacena para que en el siguiente bucle de envío sea nuevamente distribuida y procesada.

El segundo posible resultado es que no se haya devuelto ningún resultado a pesar de que todas

se hayan procesado correctamente. Esto significa que en ese bucle de envío y distribución no se ha conseguido descifrar la clave, lo que supone realizar el cálculo de las siguientes palabras auxiliares y reanudar el proceso de despliegue.

El tercer resultado es haber encontrado la clave. Cuando ocurre este hecho, la aplicación muestra al usuario el resultado y no realiza más cálculos de palabras auxiliares.

En la figura 5.1 se puede ver la arquitectura empleada hasta el momento. El *cliente* genera una secuencia de palabras auxiliares para encapsularlas en un *trabajo* y enviárselo al servidor que será el encargado de distribuir cada una de las tareas a los nodos. En este punto cada uno de los núcleos de los microprocesadores de cada nodo, obtiene un conjunto de tareas. Como cada tarea tiene una palabra auxiliar, este va generando tantas *palabras matemáticas* como estén definidas en el intervalo. Según las va generando las transforma a palabas legibles, las encripta con la codificación definida por el usuario y las compara con la clave a hackear, hasta haber encontrado una coincidencia o haber generado todas las *palabras auxiliares* posibles.

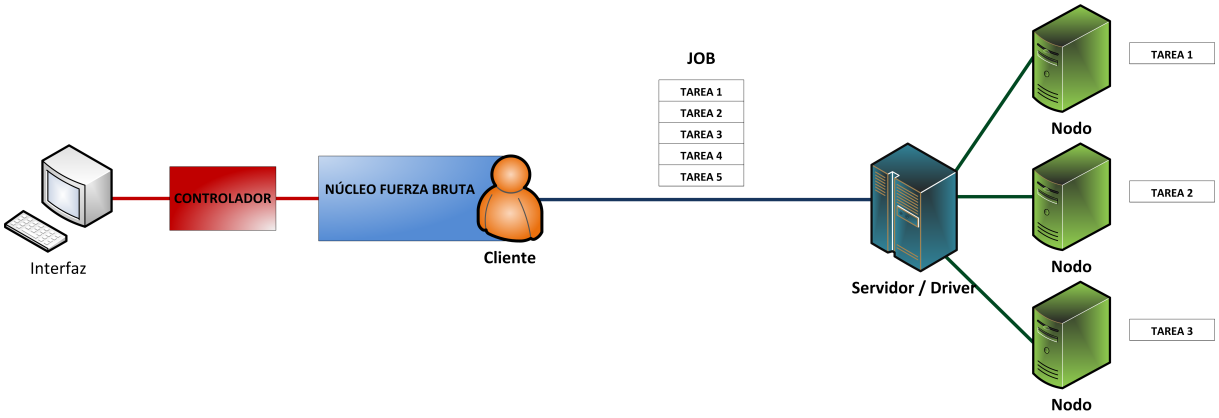


Figura 5.1: Ejemplo de una arquitectura básica de la red de DJ HACK.

5.2.2. Implantación de un proveedor de datos en memoria

Uno de los problemas que acarrea esta estructura de trabajo, es la *duplicación* de datos en cada una de las tareas como son el intervalo a usar, la clave a romper, el alfabeto, el algoritmo de encriptación a utiliza, etc.

Por lo general, más de una tarea puede ser enviada a cada nodo. Teniendo en cuenta los protocoles implementados por JPPF de comunicación y serialización; los objetos referenciados por múltiples tareas a la hora del envío, son deserializados en múltiples y distintas instancias distintas en el momento de la ejecución de una tarea en el nodo. Esto significa que, si n tareas hacen referencia al objeto A en el momento del envío del trabajo, el nodo deserializa múltiples

copias de *A*: con la tarea 1 referencia a *A*1, con la tarea 2 referencia a *A*2 , . . . , con la tarea N referencia a *An*. Como se puede ver, si el objeto compartido es muy grande, no se va a tardar en afrontar problemas graves de memoria.

Para resolver este problema, JPPF proporciona un mecanismo llamado *proveedor de datos* (*DataProvider*) que permite el intercambio de objetos comunes entre las tareas en el mismo trabajo. Un proveedor de datos es una instancia de una clase que implementa la interfaz *data-Provider*. He aquí la definición de la interfaz:

```
public interface DataProvider extends Serializable
  {
    Object getValue(Object key) throws Exception;

    void setValue(Object key, Object value) throws Exception;
  }
```

Este es de hecho una interfaz objeto de mapa: se pueden almacenar objetos y asociarlos con una clave, a continuación, recuperar estos objetos mediante la clave asociada.

Se pueden ver dos ejemplos de cómo se implementa un mapa de memoria y como tratar con su estructura:

En la aplicación

```
MiObjetoPesado objetoPesado = \ldots;
// Creacion del proveedor de datos
DataProvider dataProvider = new MemoryMapDataProvider();
//Almacenamiento del objeto en el proveedor de datos
dataProvider.setValue(miClave, objetoPesado);
//Asociacion del proveedor de memoria con una trabajo
JPPFJob = new JPPFJob(dataProvider);
job.add(new MyTask());
```

En la tarea

```
public class MiTarea extends JPPFTask
  {
    public void run()
    {
      //Se obtiene la referencia del proveedor de datos
      DataProvider dataProvider = getDataProvider();
      // Se obtiene el objeto compartido
      MyLargeObject MiObjetoPesado = (MiObjetoPesado)
          dataProvider.getValue("miClave");
```

```
        //Uso de los datos
    }
}
```

A pesar de que DJ HACK no dispone de una gran cantidad de datos comunes entre los nodos, por cuestiones de rapidez y eficiencia, primordiales al a hora de realizar ataques de fuerza bruta, se ha diseñado un mapa donde albergar todos los datos comunes de un ataque. Posteriormente este mapa se incluye en cada *job* a procesar para que los nodos compartan el mismo *alfabeto*, el mismo *intervalo* y el mismo *algoritmo de encriptación*. Así pues las tareas que se ejecuten en dichos nodos dispondrán de los datos requeridos para realizar el ataque.

5.2.3. Análisis de problemas

El uso exclusivo de JPPF limita en gran parte a DJ HACK, es decir, cada vez que se envía un trabajo al servidor hay que esperar hasta que estén procesados todos los resultados para obtenerlos y poder determinar si se ha encontrado la clave o hay que seguir generado nuevos *jobs* con un nuevo conjunto de *palabras auxiliares.*

Esto supone un serio problema en el rendimiento de la aplicación. Cuando una tarea es ejecutada en un nodo y consigue decodificar la clave encriptada, los demás nodos no son conscientes de ello y continúan realizando su ataque hasta acabar con todas las tareas pendientes que hay en la cola del servidor. Esto conlleva que se tengan que ejecutar siempre todas las tareas del trabajo, lo cual ralentiza bastante el proceso de hackeao pues JPPF no permite parar la ejecución de un trabajo al cumplirse una determinada condición o realizar una intercomunicación nodal, es decir, comunicar los nodos entre sí.

Para poder subsanar este inconveniente se ha utilizado otro framework llamado Hazelcast 4.1.3.

5.2.4. Implantación de Hazelcast y clases de inicialización

La problemática de la intercomunicación nodal, acarrea el tener que esperar, una vez enviado un trabajo, hasta que se finalice completamente. Por ello la arquitectura requiere de una ampliación trascendental.

La idea de DJ HACK ha sido, a través del framework de Hazelcast y utilizando el método *Hazelcast.getMap(String nombreMapa)*, desarrollar otra estructura de datos compartida por todos los nodos, donde cada uno de los componentes se unen como oyentes. La finalidad del uso del Hazelcast es bien sencilla:

Cuando una tarea que se está ejecutando encuentra la clave a decriptar, esa misma tarea añade un *booleano* al mapa de hazelcast a través del método *HazelcastMap.put(String Condicio-*

nAtomica, AtomicBoolean conditionReached) y finaliza su ejecución. Las demás tareas que se estén ejecutando al estar como oyentes al mapa mediante el método *Hazelcast.addEntryListener (Object clase, Boolean esOyente)*, detectan rápidamente que el mapa dispone del objeto que determina que ya se ha encontrado la clave y que no es necesario seguir con el procesamiento de claves. Así pues, todas las tareas que faltan por procesar, cuando lleguen a cada uno de los nodos, revisan primeramente si se encuentra algún dato dentro de la memoria compartida, y si es así, finalizan su ejecución sin realizar ningún tipo de ataque.

El principal problema que supone esto, es albergar de manera continuada el mapa en la memoria en los nodos, pero para ello se ha desarrollado una librería llamada *StartupClasses.jar*. Esta librería contiene todas las clases que se van a ejecutar en cada nodo a la hora de su inicialización.

Las *clases de inicio* permiten que un pedazo de código se ejecute en el tiempo de inicio de un nodo o servidor. Se pueden utilizar para muchos propósitos, incluyendo la inicialización de los recursos, conexiones a la base de datos, etc. Las *clases de inicio* se definen usando la *interfaz del proveedor de servicios*. El flujo de trabajo general para generar una clase personalizada de inicio es el siguiente:

Crear una clase que implementa la interfaz de la clase del proveedor de inicio. Para hacer que la clase de inicio sea procesada por el nodo, se necesita que la clase implemente la interfaz *JPPFNodeStartupSPI*, que proporcionará al nodo la información suficiente para crear y ejecutar la *clase de inicio*. Esta interfaz se define de la siguiente manera:

```
public interface JPPFNodeStartupSPI extends JPPFStartup { }
```

Como se puede ver, esto es sólo una *interfaz de marcador*, usada para distinguir entre el inicio de clases en un nodo y las *clases de arranque* del servidor. La interfaz del proveedor para la inicialización de un nodo o un servidor extiende de la interfaz *JPPFStartup*, donde esta extiende de *java.lang.Runnable*. Por tanto, escribir una clase de inicio consiste esencialmente en la escritura del código en el método *run()*. En el siguiente código se puede ver en una aplicación que simplemente imprime un mensaje cuando se inicia el nodo:

```
package org.jppf.example.startup.node;

\begin{lstlisting}
  import org.jppf.startup.JPPFNodeStartupSPI;

  public class TestDeInicio implements JPPFNodeStartupSPI
  {
    public void run()
    {
      System.out.println("Soy la clase de inicio en un nodo");
```

```
    }
  }
  \
```

5.2.4.1. Crear el archivo de definición de servicio

Para que las *clases de inicio* sean detectadas por JPPF, se debe crear, en la carpeta raíz, un directorio llamado *META-INF/services*. En este directorio es necesario crear un archivo llamado *org.jppf.startup.JPPFNodeStartupSPI.* Y ese archivo debe de estar definido con la ruta completa de la clase de inicio.

```
org.jppf.example.startup.node.TestDeInicio
```

5.2.4.2. Despliegue de la clase de inicio

Ahora sólo queda crear un archivo *jar* que contenga todos los artefactos que se han creado anteriormente: la *clase de inicialización* junto con la carpeta *META-INF/services*. Posteriormente una vez obtenido el jar este debe de ser agregando a la ruta de la clase; bien del servidor, si queremos que todos los nodos conectados al servidor utilicen la clase de inicialización; o agregarlo a la ruta del nodo, si sólo se requiere un único nodo para su uso.

Una vez conocidos estos conceptos, el flujo de trabajo de DJ HACK queda más definido. Cuando se inicializa un nodo, este se conecta al servidor, el cual le envía de manera serializada la clase encargada de inicializar los nodos. Esta clase permite al nodo unirse como oyente al mapa distribuido de hazelcast (o en el caso de ser el primer nodo en inicializarse crea el mapa y se une como oyente), así como guardar una instancia del mapa en memoria. Hay que recordar que el mapa de Hazelcast no se encuentra en un único nodo; sino que según se van inicializando los otros terminales, la estructura se va fragmentado por cada uno de ellos, consiguiendo así una jerarquía igualitaria donde cada nodo alberga una parte del mapa. Por consiguiente, en el caso de que algún nodo se caiga, la estructura se mantiene intacta en los demás terminales de clúster.

Una vez los nodos estén inicializados, se puede proceder al ataque. Cada una de las *tareas* desplegadas a la hora de enviar un *trabajo* al servidor, en sus primeras líneas de ejecución accede al mapa distribuido de Hazelcast ubicado en la memoria del nodo. Su acceso es relativamente sencillo, pues la clase de inicialización sigue un patrón síngleton. Una vez obtenido el mapa, los nodos se unen como oyentes para asegurarse de que si en algún momento otra tarea en otro nodo ha encontrado la clave, estos podan finalizar su ejecución lo antes posible. Posteriormente, se procede a realizar el hackeo. En el momento en el que una tarea encuentre la clave, esta ubica en la estructura de hazelcast el *booleano atómico* que será detectado por todos los oyentes al mapa, así pues, todas las tareas que se estén ejecutando en ese preciso instante detectan el cambio en

el mapa y finalizan su ejecución instantáneamente. Por otro lado, gracias a la posibilidad de albergar el mapa en la memoria de cada uno de los nodos, las tareas que estén pendientes de computo, se ejecutarán rápidamente en los nodos sin realizar ningún tipo de acción salvo la de comprobar que existe dicho booleano determinativo en el mapa.

Finalmente los resultados son devueltos al cliente que se encargará procesarlos.

Con esta metodología y gracias a los framewoks de JPPF y Hazelcast se consigue subsanar el problema de tener que esperar a que se ejecuten todas las *tareas* del *trabajo* de manera completa aun habiéndose encontrado la clave.

5.2.5. Implantación de un pool de conexiones

Aun habiendo utilizado JPPF con Hazelcast para optimizar el rendimiento y velocidad del ataque de fuerza bruta. Existe otro problema muy importante, pero para poder explicarlo es necesario tener en cuenta unos conceptos básicos.

Cuando un *trabajo* es enviado al servidor, este despliega sus tareas en los nodos. Si algún nodo se cae de la red, atreves de los mecanismos proporcionados por JPPF el driver se encarga de distribuir sus tareas entre los demás nodos consiguiendo no perder ninguna tarea por procesar. Esto sigue siendo efectivo tanto si se cae un nodo, como si se cae la red entera ya que las tareas siguen encoladas a la estructura del *driver*, con lo que no se pierde absolutamente ningún *trabajo*.

Uno de los problemas a tener en cuenta, es la caída de todos los nodos de la red. Una caída tan radical, eliminaría por completo cualquier rastro del mapa distribuido de Hazelcast, ya que este está compuesto por cada uno de los nodos de la aplicación. La caída del mapa distribuido supone un peligro muy grande para la aplicación, ya que si una de las tareas ha encontrado la clave y posteriormente todos los nodos caen, al volver a inicializar los nodos se crea un nuevo mapa distribuido, con lo que las siguientes tareas no tendrán constancia de que la clave ya ha sido encontrada y seguirán realizando el ataque de fuerza bruta en vez de finalizarse instantáneamente.

Para poder subsanar este problema, se han realizado conexiones por cada nodo a una base de datos. Así pues, en cuanto una tarea encuentra la clave, aparte de insertar un *booleano* determinativo en el mapa de Hazelcast, se inserta un registro en la base de datos. De esta manera cuando todos los nodos caen, el mapa compartido queda totalmente inservible, pero a las tareas les queda como referencia el registro de la base de datos para saber si deben continuar o no con el ataque.

Por ello se implementaron diferentes metodologías para realizar las conexiones, a continuación se muestran las técnicas empleadas y sus consecuentes problemas:

Primeramente se intentó que por cada tarea se abriese y cerrase una conexión, esto fue

totalmente catastrófico pues existen numerosas tareas que procesar y ralentizaba en gran medida la rapidez del ataque.

Como segunda opción, y atreves de la *clase de iniciación* la idea era realizar una única conexión al inicializarse cada nodo donde las tareas pudiesen utilizarla. Esto no ocasiona ningún problema en procesadores mono hilo, ya que es el único núcleo el que utiliza la conexión. Pero debido a que hoy en día la mayoría de los procesadores son multinúcleo, se ejecutan de manera paralela más de una tarea y una única conexión por nodo a la base de datos para más de una solicitud no es viable. Esto quiere decir, que mantener una única conexión abierta compartida, acarrea problemas de concurrencia de hilos, sobre todo si varios hilos intenta hacer una operación con la conexión sin sincronizarse entre ellos. Y realizar una sincronización de hilos ralentizaría mucho el ataque.

Para evitar los problemas de concurrencia, lo mejor es no abrir directamente la conexión con el *DriverManager*, sino delegar esta tarea en una clase que implemente la interface *javax.sql.DataSource* y, por supuesto, elegir una implementación adecuada.

Una implementación interesante de estos *DataSource* son los *pool de conexiones*. Básicamente, estos *pools* nos facilitan conexiones según se van pidiendo, pero las *reaprovechan* de una petición a otra. La idea es la siguiente:

Se pide al *pool* una conexión. Este busca una que esté libre y la devuelve, apuntando que está en uso y que deja de estar libre. Después se realiza la operación (consulta, inserción, borrado, ...) se cierra la conexión. El *pool* recibe esta petición de cierre y no cierra la conexión, sino que la deja abierta y la vuelve a marcar libre para el siguiente que la pida.

Por ello, se introduce en DJ HACK el mecanismo encargado de realizar un *pool de conexiones* [4] a través de la clase *BasicDataSource* de *apache commons-bdcp* a la base de datos, ya que en la API de java no existe implementación alguna de este *DataSource*. Este *pool* es además configurable para que compruebe si la conexión es correcta antes de servirla al que se la pida, para que las verifique cada cierto tiempo, etc.

Para realizar la conexión en DJ HACK es necesario incluir en el *jar* de *inicialización*, la configuración de la base de datos (usuario, contraseña, dirección IP de la máquina donde se está ejecutando la base de datos) y teniendo en cuenta los ficheros de configuración del nodo donde se especifica con cuantos hilos en paralelo se quiere trabajar, cada nodo al inicializarse crea tantas conexiones a la base de datos como hilos se vayan a utilizar para el ataque. De esta manera cuando varias tareas requieren de una conexión a la base de datos de manera simultánea, siempre disponen de una conexión libre para acceder a la base de datos, evitando cualquier problema de concurrencia.

[4] Véase el pool de conexiones en la sección 3.5

5.2.5.1. Análisis de problemas

Una vez realizado un ataque, los registros añadidos a la base de datos quedan modificados, así como los objetos que pueda haber en el mapa distribuido. Con lo cual si se quiere realizar un nuevo ataque, seguramente habrá problemas, ya que si en el primer ataque la contraseña ha sido hallada, el segundo ataque lo detecta y no realiza ninguna acción pues piensa que la clave ya ha sido encontrada.

Para ello, se ha diseñado una *tarea centinela*[5], encargada de establecer los valores por defecto del mapa distribuido y de la base de datos.

5.2.6. Resumen

En este capítulo se explicado paso a paso el despliegue de un ataque de fuerza bruta. Desde el envío de un trabajo, hasta las conexiones necesarias para poder realizarlo de manera segura y eficaz.

La idea es bastante compleja, lo primero antes de realizar un ataque es inicializar tanto los servidores como los nodos. El servidor dispone un archivo *jar* que es desplegado a cada uno de los nodos en su inicialización. La clase de inicialización que se ejecuta en cada uno de los nodos es la encarga de conectarse al mapa distribuido (o crearlo en el caso de no existir) que comparten todos los nodos. Además esta clase inicializa un pool de conexiones a la base de datos para ser usadas por cada uno de los hilos del procesador. Cada vez que un nodo se inicializa, comprueba si el mapa distribuido al que se ha conectado dispone de algún dato. Si el mapa dispone de algún dato, significa que se ha encontrado la clave al decriptar, luego todas las tareas que se ejecuten en ese nodo no realizan ningún hasheo sino que finalizan rápidamente. En el caso de que el mapa no disponga de ningún dato, se realiza un pool de conexiones con la base de datos y se comprueba en el registro si se ha encontrado la clave o no. De no haberse encontrado la clave, se procede a la ejecución de las tareas. No obstante, el mero hecho de existir un registro en la base de datos y no disponer de ningún dato en el mapa, significa que la clave ha sido encontrada, que todos los nodos se han caído, y que es el primer nodo que se inicializado después de la caída. Por ello, se inserta un *booleano* determinativo en el mapa de hazelcast, para que los nodos que se van a conectar posteriormente detecten que la clave ha sido encontrada.

Una vez obtenida la inicialización de la red, se envía primeramente, a través del *cliente*, una tarea centinela para limpiar las modificaciones posibles que se hayan podido producir en ataques anteriores.

A continuación se puede empezar a realizar el hasheo. A través de los datos primordiales (número de palabras auxiliares a generar en cada bucle de envío, número de palabras matemáticas a generar, la clave a decriptar, algoritmo a usar, etc.) proporcionados por el usuario, se

[5] Véase el código completo en la apéndice B

generan una cantidad de palabras auxiliares que son encapsuladas en un *job*. Y al *trabajo* se le añade un proveedor de memoria con los datos comunes a todas las tareas.

Posteriormente, se realiza la conexión a través del *cliente* con los servidores disponibles y se procede a realizar el envío del trabajo. Primeramente el *proveedor de memoria* es desplegado por los servidores disponibles a sus nodos conectados, y posteriormente, se procede al envío de trabajo. Las tareas del trabajo quedan almacenadas en la cola del servidor encargado de distribuirlas por todos los nodos.

Cuando una tarea es ejecutada por un nodo, la primera acción que realiza es comprobar si el mapa de memoria de Hazelcast dispone de algún dato, en el caso de que exista finaliza su ejecución, no obstante si el mapa no contiene ningún elemento procede a realizar el ataque.

Una vez obtenidos los resultados del bucle de envío, se comprueba si se ha obtenido la contraseña para mostrar los resultados al usuario y finalizar el ataque, pero de no ser así, la aplicación genera la siguiente secuencia de *palabras matemáticas* para volver a distribuir; hasta haber encontrado la clave, o no existan más palabras matemáticas de una determina longitud.

Para tener una visión general de la arquitectura de la red véase la figura 5.2:

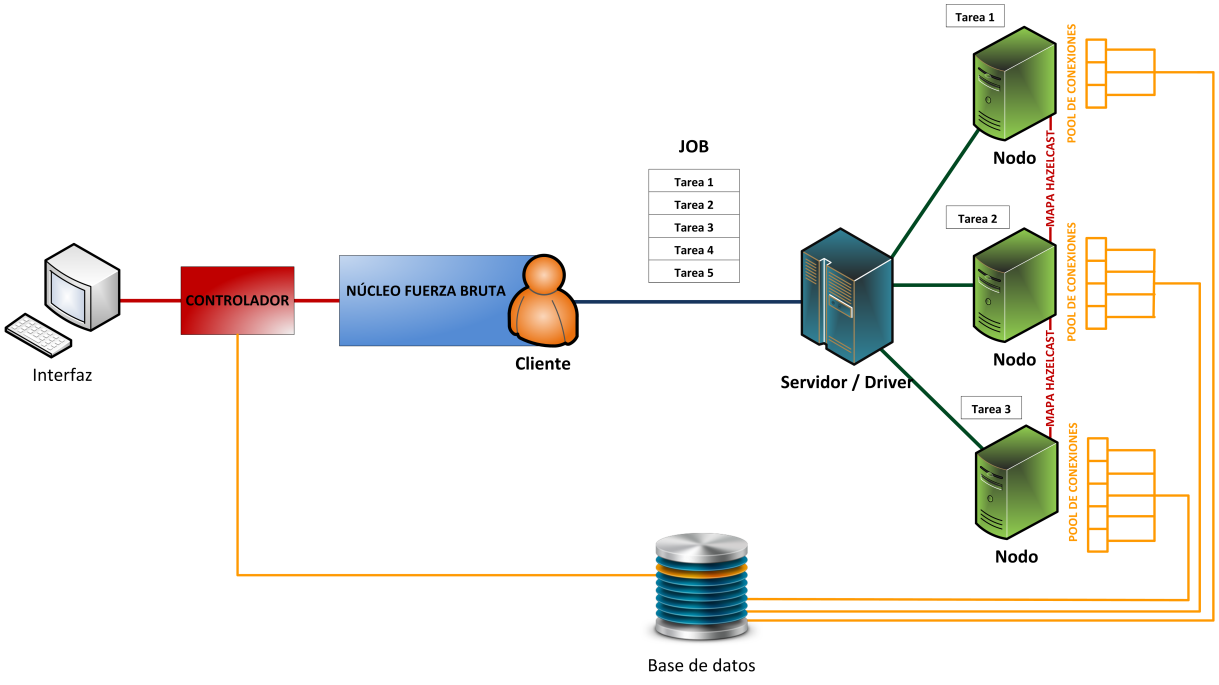


Figura 5.2: Ejemplo de una arquitectura completa de la red de DJ HACK.

5.3. Objetivo 3: Realización de un sistema de backup

Cuando se está realizando un ataque de fuerza bruta pueden surgir problemas como una caída de los servidores o un fallo en el sistema, además el tiempo que se tarda en realizar un ataque depende siempre de la complejidad del mismo. Un alfabeto con demasiados caracteres a procesar, una capacidad de procesamiento bajo de cada uno de los componentes del clúster, la posible saturación del tráfico en la red... pueden ocasionar que el ataque sea lento, lo que implica tener que tener en marcha todos los dispositivos de una manera ininterrumpida hasta la finalización del ataque. Para poder subsanar este problema DJ HACK ha desarrollado un sistema de backups para la fuerza bruta.

Cuando se inicia un ataque de fuerza bruta, el usuario especifica cual es la clave a romper, el diccionario a utilizar, los posibles tamaños que puede llegar a tener la clave decriptada, etc. Todos esos registros quedan almacenados en una base de datos, con el fin de reutilizarlos posteriormente. A través de esos datos, el sistema empieza a generar *palabras auxiliares* de un tamaño determinado, de no haberse encontrado la clave con ese tamaño, continua con el siguiente tamaño de palabra especificado por el usuario, y así, sucesivamente hasta haber procesado todas las palabras de todos los tamaños definidos por el usuario, o haber encontrado la clave.

Como se ha comentado anteriormente, esta tarea puede llegar a durar un tiempo indeterminado. Cualquier fallo que pueda ocurrir ocasionaría la perdida de todo lo procesado hasta ese momento y se tendría que volver a comenzar de nuevo. Por ello, la aplicación dispone del sistema de *backups*.

Cuando se envía a procesar por primera vez una clave nueva, el núcleo de la fuerza bruta genera un conjunto de palabras auxiliares para una longitud determinada. En este primer bucle el conjunto de palabras auxiliares son enviadas al servidor encapsuladas en un job encargado de distribuirlas por los nodos. Tras acabar con el proceso de hackeo de ese primer conjunto, el núcleo empieza a analizar los resultados con la finalidad de determinar si se ha encontrado la clave. De no ser así, se procede a la realización del *backup*.

El sistema de *backups* se basa en un almacenamiento progresivo en la base de datos de *palabras auxiliares*. Una vez procesados los resultados, en cada bucle de envío de tareas al servidor, en el caso de no haber ningún error en alguna de las tareas, el sistema se encarga de almacenar la última *palabra auxiliar* procesada, logrando de esta manera tener una referencia de las palabras que se han computado. No obstante, de ocurrir algún problema y debido a las copias de seguridad únicamente se realizan cuando se obtienen los resultados de un determinado bucle de envío, tanto las tareas que estén pendientes de procesar en la cola del servidor como las que estén procesándose en los nodos se pierden. En el código que se muestra a continuación

se puede ver el método encargado de realizar el backUp:

```
/**
     * <p>Realiza un backup en la base de datos con la ultima palabra
         procesada correctamente.</p>
     *
     * @throws SQLException
     * @throws IOException
     */
    public void realizarBackUp() throws SQLException, IOException {

        //Se convierte la palabraAuxiliar en una cadena legible para
           MySQL
        String palabraAuxiliarSQL = "";
        int caracter;
        for (int i = 0; i < palabraAuxiliar.length; i++) {
            caracter = palabraAuxiliar[i];
            //En el ultimo caso no se anade la coma del final
            if (i == palabraAuxiliar.length - 1) {
                palabraAuxiliarSQL = palabraAuxiliarSQL.concat("" +
                   caracter);
            } else {
                palabraAuxiliarSQL = palabraAuxiliarSQL.concat(
                   caracter + ",");
            }
        }
        //Se almacena en la base de datos
        Connection con = SeguridadBaseDatos.getSeguridadBaseDatos().
           conectarBD();
        Statement st = con.createStatement();
        st.executeUpdate("Update shadecrypter.claves set backUp='" +
           palabraAuxiliarSQL + "'where claveHash='" +
           palabraEncriptada + "' "
                + "and idEncriptacion='" + idEncriptacion + "';");

    }
```

A través de este sistema, se consiguen dos funcionalidades elementales para la aplicación:

La primera es dotar a DJ HACK de un sistema seguro, donde cualquier problema interno o externo no ocasione la pérdida total del trabajo realizado en un ataque de fuerza bruta.

La segunda funcionalidad, es la posibilidad de parar el ataque cuando el usuario lo desee, sin

perder todo el trabajo de cómputo realizado hasta ese momento. De esta manera, el usuario podrá reanudar el hackeo cuando él lo crea conveniente. Consiguiéndose así, un sistema más dinámico donde es el usuario el encargado de iniciar, reanudar, cancelar un ataque cuando lo crea necesario.

5.4. Objetivo 4: Sistemas de seguridad

Con el fin de proteger todos los datos y acciones realizadas por el usuario en la aplicación, DJ HACK proporciona una serie de sistemas de seguridad. La aplicación está diseñada para interactuar con la base de datos de una manera constante; para poder realizar las conexiones, se requieren los datos de acceso a dicha base, como son el nombre usuario, la contraseña y la dirección donde está ubicada la base de datos.

Está configuración está contenida en el directorio de la aplicación, pero además debe transmitirse a través de la red para que los nodos puedan hacer uso de ella. Esto acarrea un problema de seguridad grave, pues cualquier individuo que acceda físicamente al computador donde se está ejecutando la aplicación o realice ataques de *sniffing* puede capturar estos datos y utilizarlos con fines malintencionados.

Para poder subsanar este problema, se han implantado varios sistemas de seguridad, uno a nivel local y otro a nivel de red.

5.4.1. Seguridad a nivel local

Cuando un usuario accede al sistema, es necesario que configure los parámetros necesarios para conectarse con la base de datos. Estos datos se almacenan en el fichero *Database.properties* para que en posteriores accesos a la aplicación no se requiera realizar nuevamente la configuración. Cualquier persona que pueda acceder a ese fichero, podría ver la configuración de la base de datos y podría acceder a ella sin ningún problema. Por ello, se ha tenido que cifrar la información contenida en dicho fichero.

Para poder acceder a la aplicación se requiere el nombre de usuario y contraseña. Estos datos se utilizan, aparte de para la autenticación de usuario a la aplicación, para encriptar los datos de acceso a la base de datos. La metodología empleada para realizar las encriptaciones es la siguiente:

Cuando un usuario inicia la aplicación por primera vez, debe autenticase en el sistema con el nombre de usuario y la contraseña. Estos dos datos son concatenados junto a un *grano de sal* de diecinueve caracteres para formar la contraseña real que se utiliza en la aplicación. Con lo cual la contraseña a cifrar queda definida de las siguiente forma: *usuario + contraseña +grano de sal*.

El *grano de sal*, está compuesto por números, letras en mayúscula y minúscula y caracteres especiales. La finalidad del *grano de sal* es aumentar el tamaño de la contraseña para que las encriptaciones sean mucho más robustas.

A una vez obtenida la contraseña, el siguiente paso a dar es codificarla, para ello se ha hecho uso de la librería *Jasypt*[6] que proporciona los métodos necesarios y suficientes para realizar la encriptación de la contraseña.

La codificación se realiza a través de los *digestores* proporcionados por *Jasypt*. Los *digestores* son clases especializadas en la creación de *resúmenes* de mensaje de entrada, en otras palabras, son los encargados de realizar la codificación hash. El *digestor* empleado en la codificación de la contraseña es el *StandardStringDigester* de *Jasypt*. Se ha utilizado este digestor ya que soporta una serie de configuraciones que hacen que la clave hash generada sea aún más segura. A continuación, se detallan y especifican, los pasos realizados para encriptar la configuración del sistema que se usa, tanto para la autenticación de usuario, como para la encriptación de la configuración de cada uno de los datos de la base de datos.

Inicialmente se determina y se construye el *digestor* a utilizar, en este caso, se ha usado el constructor *StandardStringDigester()*. Para utilizar este *digestor* se han tenido que configurar los siguientes parámetros:

Mediante el método *setAlgorithm(String algoritmo)*se establece el algoritmo a utilizar en el hasheo. La aplicación utiliza SHA-256 como algoritmo de codificación base.

Y para proporcionar más seguridad a la codificación, se ha utilizado el método *setIterations(String iteraciones)*. Definiendo este parámetro, se consigue que la contraseña sea hasheada tantas veces como esté definido en iteraciones. DJ HACK utiliza 1302 iteraciones, para la realización del hashing de su contraseña de acceso al sistema.

Finalmente, una vez obtenida la clave encriptada, esta queda almacenada en el fichero *security/keyStore*.

En la figura 5.3 se puede ver un esquema básico empleado en la realización en la encriptación:

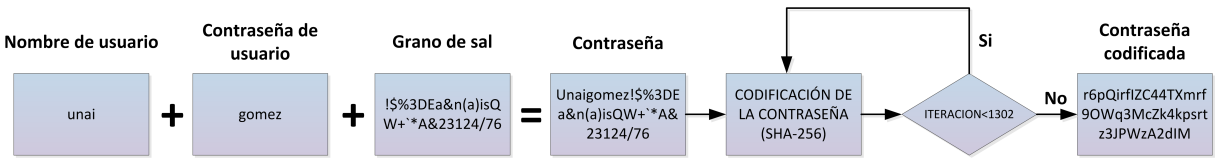


Figura 5.3: Esquema básico de encriptación.

[6]Más información sobre *Jasypt* en la sección 4.1.4

Utilizando este sistema de seguridad, se ha logrado un acceso muy seguro a la aplicación.

Por otro lado, se ha codificado la configuración de acceso a la base de datos. El método empleado, es totalmente diferente al usado en la codificación del sistema. Las encriptaciones realizadas en la en la configuración de la base de datos, siguen la especificación *PBE* (*Password Based Encription*), pues no es posible realizar encriptaciones a través de *digestores*, ya que estos realizan codificaciones unidireccionales. Es decir, a través de un *digestor* se puede hashea una contraseña, pero esa palabra hasehada no puede ser revertida para obtener la contraseña real. Debido a esto, es necesario el uso de herramientas que realicen codificaciones *PBE*, ya que estas codificaciones son bidireccionales, es decir, a través de la contraseña proporcionada por el usuario se puede codificar y decodificar un elemento. En DJ HACK se utiliza una codificación *RSA*[7] para encriptar la configuración de acceso a la base de datos. La metodología empleada para esta encriptación es la siguiente:

Una vez construido en encriptador *PBE* a través de su constructor *BasicTextEncryptor()*, se determina la clave a utilizar para realizar las encriptaciones mediante el método *setPassword(claveCompuesta)*. En la aplicación, se ha usado la clave compuesta formada por el nombre de usuario + contraseña de acceso + el grano de sal, que se ha determinado anteriormente por el sistema de autenticación de la aplicación.

Seguidamente, utilizando los métodos *encrypt(String texto)* y decrypt(String texto) se codifica el nombre de usuario, contraseña y dirección de acceso a la base de datos.

Finalmente, para poder preservar la configuración realizada por el usuario de manera segura, una vez obtenida la configuración encriptada, esta queda almacenada en el fichero *config/Database.properties*.

5.4.2. Seguridad a nivel de red

Cuando se ejecuta un ataque de fuerza bruta, el driver despliega el paquete *Startup.jar* en cada uno de los nodos. En ese paquete, se encuentra ubicado el fichero de configuración de la base de datos para que los nodos puedan realizar un *pool de conexiones*. Debido a que la encriptación de la configuración de la base de datos sigue la especificación PBE, los nodos no disponen de la contraseña requerida para decodificar la configuración y realizar la conexión a la base de datos. Esto supone un problema muy grave, ya que se le obliga al servidor enviar esta configuración decriptada, con la consecuencia de que si la red está construida bajo un entorno no seguro donde cualquier persona puede acceder a cualquier terminal físicamente, cualquier individuo podría interceptar la configuración a través de técnicas de *sniffing* y acceder libremente a la base de datos. Debido a esto DJ HACK incorpora dos mecanismos de encriptación

[7] Véase la sección 3.3.4.3

a nivel de red: la encriptación de paquetes a través de JPPF[8] y la encriptación SSL[9] para las conexiones con la base de datos.

Encriptación de paquetes a través de JPPF:

En las trasmisiones realizadas por JPPF entre el cliente, servidores y nodos, la mayor parte del tráfico de la red está hecha de objetos Java serializados. De forma predeterminada, estos objetos serializados son enviados a través de la red sin ningún tipo de ofuscación o cifrado. Esto puede ser considerado de riesgo en entornos de alta seguridad, como lo es DJ HACK. Para mitigar este problema, JPPF proporciona un *add-on* que permite la transformación de un bloque de datos en otro bloque de datos, así como la transformación revertida.

Para entender mejor cómo funciona este mecanismo, primero tener una visión general de cómo los componentes de JPPF(cliente, servidores y nodos) envían y reciben mensajes a través de la red. Un mensaje en JPPF se compone de un número de bloques de datos, cada bloque que representa un objeto serializado (o un objeto gráfico) precedido por su longitud.

L1 Bloque1 ….. Ln BloqueN

Dónde:

- Bloque1, . . . , BloqueN son bloques separados de datos que constituyen el mensaje.
- L1, . . . , Ln son las longitudes de cada bloque de datos.

La herramienta de transformación de datos permite al desarrollador transformar cada bloque de datos. El uso de la interfaz *JPPFDataTransform* incorpora los métodos *wrap(InputStream source, OutputStream destination)* y *unwrap(InputStream source, OutputStream destination)* encargados de transformar los bloques.

La longitud de cada bloque siempre es computada por JPPF. Por ejemplo, si la transformación de datos utilizada es una forma de cifrado (y descifrado para la operación inversa), entonces todo menos la longitud de un bloque se encripta.

Utilizando los ejemplos proporcionados por JPPF, los datos de la red pueden ser cifrados mediante un algoritmo de cifrado *DES*[10] de 56 bits con una clave secreta simétrica.

La clave secreta inicial encargada de cifrar los datos, se guarda en un almacén de claves, que se incluye en el fichero *jar* resultante que junto al fichero *StartupClasess* se deberá desplegar en los nodos, servidores y clientes. Esta clave no se utiliza para cifrar o descifrar realmente los datos. En cambio, si utiliza para generar y cifrar un duplicado de la llave secreta para cada nuevo bloque de datos para cifrar.

[8] Véase la sección JPPF en el apartado 4.1.2

[9] Para más información sobre SSL visite el apartado 3.3.6

[10] Véase la sección DES en el apartado 3.3.4.4.

Esto significa que cada bloque de datos (es decir, una *tarea* o un *proveedor de datos*) se cifra con una clave diferente. La estructura del bloque resultante es el siguiente:

- La longitud de la nueva clave.
- El contenido de la nueva clave (cifrado con la clave inicial).
- Los datos reales (cifrada con la nueva clave)

Queda, sin embargo, una vulnerabilidad: todavía se necesita la contraseña del almacén de claves. Para evitar el almacenamiento de la contraseña en claro, se ofusca utilizando una codificación *Base64* a través de la biblioteca *iHarder.net*. La contraseña ofuscada se almacena en un archivo, que también se incluye en el archivo *jar* a desplegar.

De esta manera, ubicando el archivo *jar* en cada uno de los componentes del clúster y descomentando la propiedad que se ve a continuación en cada uno de los componentes, se realiza la codificación de la transmisión entre cliente, servidores y los nodos.

```
jppf.data.transform.class = org.jppf.data.transform.DESCipherTransform
```

5.5. Encriptación SLL para base de datos

A pesar de disponer una encriptación de las transmisiones entre el cliente, nodo y servidor, las comunicaciones por parte de la aplicación, como el pool de conexiones que realiza cada nodo a la base de datos no están cifradas; con lo cual, al realizarse cualquier conexión con la base de datos, se pueden interceptar las configuraciones de la conexión (nombre de usuario, contraseña, dirección IP).

Para poder solventar este problema, la aplicación dispone de soporte para comunicaciones seguras a través de SSL. Suponiendo que la base de datos *MySQL* se ha configurado para la compatibilidad con SSL, es muy fácil comunicarse de forma segura ella.

La comunicación a través de SSL se puede lograr simplemente pasando una propiedad de conexión en la *URL JDBC*. El controlador *JDBC* de *MySQL* también proporciona una opción para pasar por la validación de certificados. Esto es útil en los casos en que está siendo un certificado con firma utilizado.

A continuación, se ofrece un ejemplo de código que muestra cómo comunicarse con una base de datos *MySQL* con *SSL* y *JDBC*. La propiedad *useSSL = true* se añade a la *URL* de *JDBC* para tratar de comunicarse a través de SSL. La propiedad *requireSSL = true* se puede añadir al conector sólo si el servidor de base de datos compatible con SSL. La propiedad *verifyServerCerticate = false* se establece para eludir la validación de certificados.

```
import java.sql.*;

public class TestSQLSSL {

    public static void main(String[] args) {
        Connection con = null;
        try {
            String url = "jdbc:mysql://127.0.0.1:3306/sample" + "?
               verifyServerCertificate=false" + "&useSSL=true" + "&
               requireSSL=true";
            String user = "testuser";
            String password = "testpass";
            Class dbDriver = Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(url, user, password);
        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            if (con != null) {
                try {
                    con.close();
                } catch (Exception e) {
                }
            }
        }
    }
}
```

5.5.1. Resumen general

Como se ha podido ver DJ HACK dispone de un sistema de seguridad contra ataques malintencionados.

Las autenticaciones de los usuarios se realizan a través de una encriptación hash SHA-256 unidireccional, compuesta por el usuario, la contraseña y el grano de sal que proporciona una seguridad extra a la encriptación.

Por otro lado, utilizando esos parámetros del usuario, mediante la codificación RSA, las configuraciones de acceso a la base de datos quedan totalmente cifradas en la aplicación.

Debido a que los nodos no pueden disponer de la contraseña para descifrar la configuración, es necesario desplegar dicha configuración sin codificación alguna entre los miembros de la

red. Para poder subsanar este problema, todas las transmisiones que se realizan entre el cliente, el servidor y los nodos son cifradas mediante DES, consiguiendo encriptar en cierta manera el contenido de los datos que se envían por la red, entre ellos, la configuración de la base de datos.

Finamente, para terminar de codificar todas las trasmisiones, DJ HACK proporciona soporte para conexiones seguras a la base de datos a través de SSL, consiguiendo así, que las transmisiones de datos que se realizan con la base de datos queden totalmente cifradas.

A través de todas las encriptaciones realizadas en DJ HACK, se consigue tener un entorno seguro y fiable para realizar cualquier tipo de ataque de fuerza bruta.

En la figura 5.4 se pueden distinguir las diferentes encriptaciones utilizadas en cada comunicación:

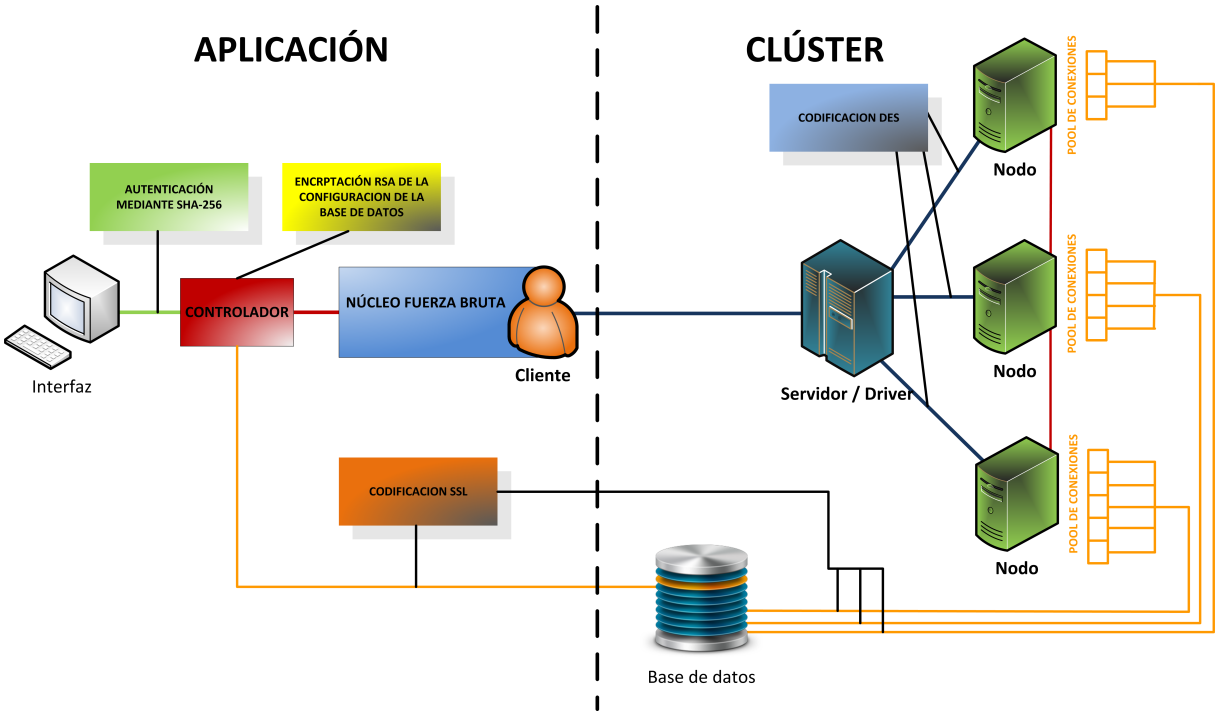


Figura 5.4: Arquitectura de la red utilizando encriptacion.

5.6. Objetivo 5: Herramientas y funcionalidades extra

DJ HACK aparte de realizar ataques de fuerza bruta de manera distribuida, proporciona al usuario una serie de herramientas muy útiles para desarrollar los ataques de una manera cómoda y automatizada, así como, la posibilidad de realizar configuraciones con un nivel de granularidad elevado.

5.6.1. Ataque por diccionario

Esta aplicación ofrece la característica de realizar ataques por diccionario.

El ataque por diccionario a diferencia de la fuerza bruta, este se ejecuta en modo local y no utiliza ningún mecanismo de cómputo en paralelo. Esto no significa que sea una desventaja para la aplicación, ya que los ataques por diccionario se basan en la lectura de un fichero compuesto por palabras, esas palabras son hasheadas y comparadas con la clave a decriptar. El tiempo empleado en este proceso, a pesar de que el diccionario disponga de una cantidad considerable le palabras, es bastante corto. Con lo cual, no es necesario realizarlo de una manera distribuida.

5.6.2. Hasheo de claves

Otra de las funcionalidades que aporta la aplicación es la capacidad de realizar hasheos propios. A través de la palabra que se quiera hashear y determinado el algoritmo (MD5, SHA1, etc.), DJ HACK encripta la palabra en el algoritmo indicado.

5.6.3. Creación automática de nodos y servidores[11]

Para poder realizar cualquier ataque de fuerza bruta, es imprescindible montar una red compuesta por servidores y nodos.

La aplicación soporta la posibilidad de crear cada uno de los componentes de una manera automatizada. Mediante de las opciones que dispone la aplicación, se pueden generar tantos componentes como el usuario desee. La herramienta aporta la funcionalidad de poder configurar[12] al detalle cada uno de los componentes de una manera sencilla a través de su interfaz.

5.6.4. Configuraciones

DJ HACK tiene la posibilidad de realizar configuraciones elementales requeridas por los ataques de fuerza bruta.

5.6.4.1. Configuración de la base de datos

Se basa en establecer los datos requeridos (usuario, contraseña, dirección IP de la ubicación de la base de datos) para realizar conexiones a la base de datos. Además, incluye la posibilidad de realizar conexiones seguras a través de SSL.

[11] La versión server no posibilita la configuración de cada uno de los componentes a través de la aplicación. Es necesario realizarla manualmente.

[12] Véase el apéndice C para ver las configuraciones.

5.6.4.2. Configuración de la fuerza bruta

En este tipo de configuración se especifican dos parámetros claves para la realización de los ataques de fuerza bruta: la cantidad de *palabras auxiliares* que va a procesar el *cliente* y la cantidad de *palabras matemáticas* que van a procesar por tarea cada uno de los nodos.

El cambio de estos valores por defecto, puede ocasionar irregularidades en el sistema si no se tienen en cuenta algunos factores:

- A la hora de establecer la cantidad de palabras auxiliares a generar, hay que tener en cuenta que un número demasiado elevado, sobrecargaría la CPU y memoria del ordenador donde se está ejecutando la aplicación, ya que este tienen que procesar tantas palabras auxiliares como estén definidas. Por otro lado, un número muy bajo, podría ralentizar el proceso de hackeo debido a una sobrecarga en el tráfico de la red, ocasionada por el constante bucle de envíos que deben realizarse. Por ello, el número de palabras a procesar por el cliente debe de ser al menos superior o igual al número de nodos disponibles, un número inferior supondría suponer que algunos nodos quedasen inoperativos, pues no habría más palabras auxiliares que distribuir.

- Por otro lado, cuando se determina la cantidad de palabras que van a procesar por tarea, se debe tener en cuenta que una cantidad demasiado elevada ralentizaría el tiempo de ejecución de cada tarea en cada uno de los nodos, incluso, dependiendo del número total de variaciones a realizar y el número de nodos disponibles podría ocurrir que algunos nodos quedasen inactivos, limitando de esta manera el computo distribuido. Por otro lado, un número muy bajo, podría ralentizar el proceso de hackeo debido a una sobrecarga en el tráfico entre los servidores y los nodos.

La combinación de estos dos factores es elemental para el proceso de hackeo; determinando, en cierta medida, la rapidez de los ataques.

5.6.4.3. Configuración del cliente

Esta configuración es esencial a la hora de realizar ataques de fuerza bruta, pues se pueden definir las direcciones IP de los servidores a los que el *cliente* se debe conectar, los puertos a utilizar en las comunicaciones, el modo de funcionamiento (*manual / automático*), la encriptación de las transmisiones entre el cliente, servidores y nodos, etc. Puede ver la configuración detallada del cliente en el apéndice C.1.

5.6.5. Consola de administración y monitorización de JPPF[13]

JPPF dispone de un sistema encargado de administrar y monitorizar todo el tráfico que hay en la red a través de una interfaz gráfica. Esta herramienta aporta al usuario las siguientes funcionalidades: la posibilidad de ver en forma de árbol todos los componentes que están conectado al cliente, el estado en el que se encuentran, que tareas están procesando, las conexiones que existen entre ellos... Además, incorpora funcionalidades a nivel de granularidad mayor; como la posibilidad de realizar acciones, como cancelar, suspender o reanudar un determinado trabajo o tarea, ver su estado, modificar las propiedades de cada uno de los componentes a distancia, etc.

5.6.6. Multiplexor de puerto TCP

El mecanismo de comunicación de la red del servidor usa TPC/IP. Para realizar su trabajo de recibir *jobs* y enviarlos a los nodos para su ejecución, son necesarios tres puertos TCP:

Un puerto está reservado para la carga de clases distribuidas, y también es utilizado por los clientes, nodos y otros servidores.

Otro puerto está reservado para la comunicación con los clientes, para recibir las solicitudes de *trabajo* y enviar los resultados.

El tercer puerto está reservado a la comunicación con los nodos, y se utiliza para enviar las *tareas* a los nodos y recibir los resultados de su ejecución. También se utiliza para delegar *tareas* en otros servidores, en el caso de que exista más de un servidor en la red.

En entornos de red en que la política impone el uso de un cortafuego, estos puertos generalmente se bloquean, impidiendo el tráfico por los puertos no autorizados. Para hacer frente a esta situación, se ha implementado una herramienta llamada *multiplexor de puerto TCP*, que permite el encaminamiento de tráfico de red desde varios puertos a un solo puerto, por un lado, y el enrutamiento del mismo tráfico desde un único puerto a múltiples, por el otro lado.

5.6.6.1. Arquitectura

Para entender cómo funciona el multiplexor y cómo se integra con una grid JPPF, se muestra el siguiente ejemplo de una configuración típica:

El cliente, servidor y nodo están todos en separados en máquinas distintas y usan los puertos por defecto 11111, 11112, 11113. Además, el entorno de la red está protegido mediante un

[13] La versión server no posibilita lanzar la consola de administración.

cortafuego, donde sólo se deja pasar el tráfico por el puerto 443. Esto da lugar, la configuración ilustrada en la figura 5.7.

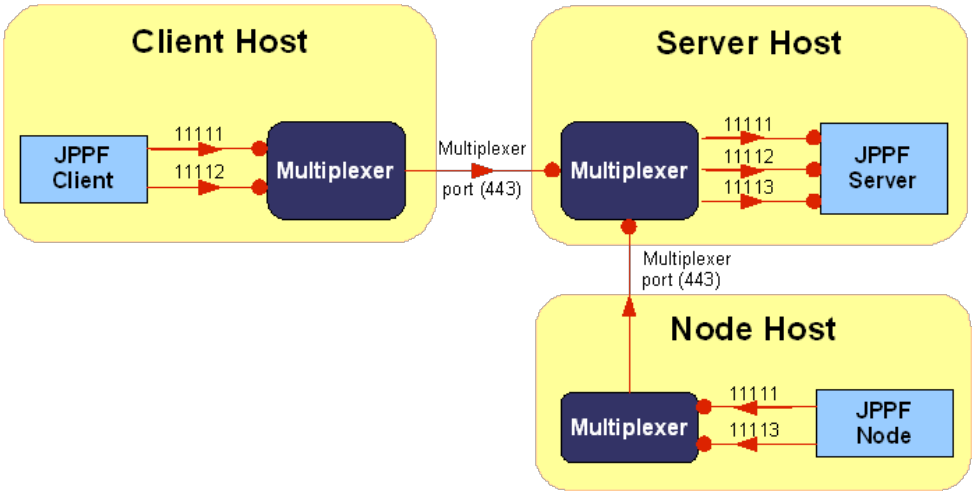


Figura 5.5: Arquitectura de la red utilizando multiplexación.

Una flecha roja dirigida representa una conexión entrante. El pequeño círculo en un extremo, significa que el componente correspondiente está a la escucha de las conexiones en el puerto especificado.

Esta arquitectura tiene una serie de observaciones: Por un lado los nodos y los clientes se conectan al multiplexor como si fuera un servidor local. Por otro lado, el servidor acepta conexiones del multiplexor como si viniesen de clientes locales o a través de los nodos. No obstante, hay que tener en cuenta, que tanto la multiplexión y demultiplexión del tráfico de la red, es completamente transparente para los componentes de la red debido en que se basa en la misma API de comunicación utilizada por JPPF, proporcionado los mismos beneficios de escalabilidad, robustez y fiabilidad. Aun así, es evidente que la utilización de este mecanismo disminuye en el rendimiento en comparación con un grid no multiplexado.

Nota: En esta configuración, la gestión y seguimiento de los nodos y del servidor no es posible, y debería de deshabilitarse para todos los nodos y servidores que estén detrás del cortafuegos.

5.6.6.2. Configuración

En términos de configuración, se ha visto en la sección de arquitectura 5.6.6.1 que el multiplexor juega un diferente papel dependiendo de si está en la máquina del servidor o no.

Un multiplexor en la máquina donde se está ejecutando el servidor, solo necesita escuchar las conexiones en un puerto de multiplexión. En cambio, un multiplexor en un cliente o nodo, se une a los puertos como si se tratara de un servidor. Por ello, se deben de especificar cuáles son

los puertos de entrada al múltiplexador y cuál es el puerto de salida por el que el múltiplexador debe enrutar el tráfico. En la figura 5.6 se muestra un ejemplo de configuración básico:

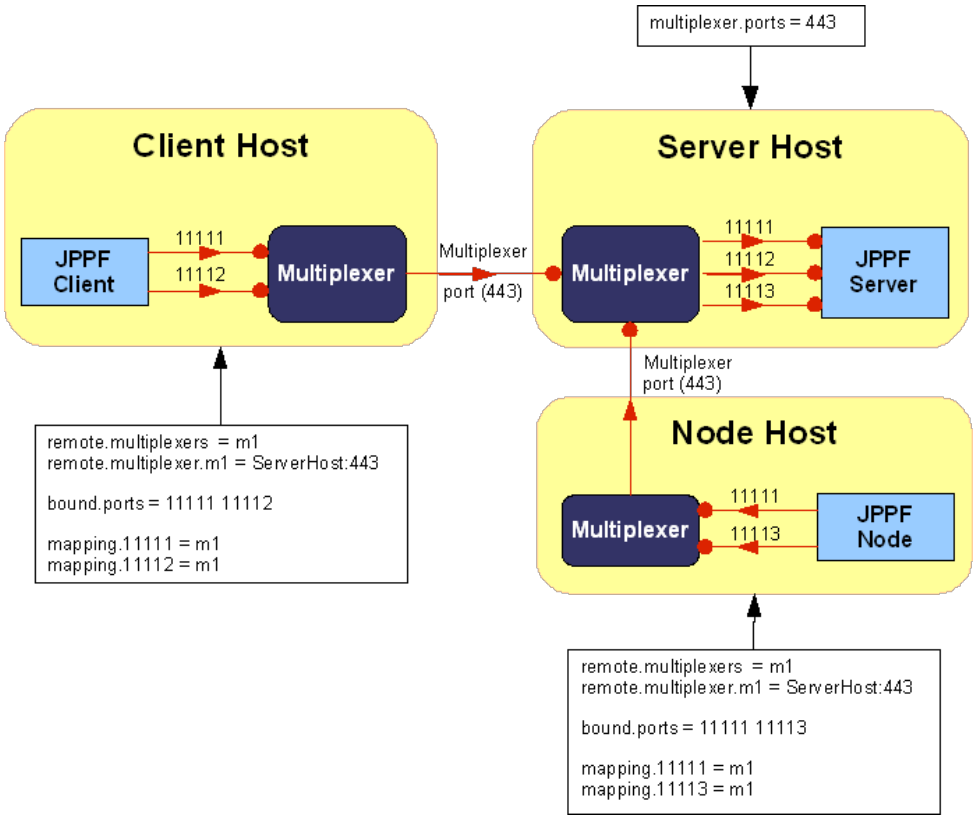


Figura 5.6: Configuración de la red utilizando multiplexación.

Configuración del servidor:

```
# Local forwarding port this multiplexer listens to
# The local port number to which to forward messages to is the
   first message
# sent by any inbound connection
 multiplexer.ports = 443
```

Configuración de nodos y clientes:

```
# Names of the remote multiplexers to connect to
 remote.multiplexers  = m1

 # Host and port corresponding to the named multiplexer
 remote.multiplexer.m1 = server_host:443

 # Local ports this multiplexer listens to
 bound.ports = 11111 11112 11113
```

```
# Mapping of the bound ports to remote multiplexers (1 per port
    )
# Communications with the bound ports will be forwarded to and
    from the
# multiplexer
mapping.11111 = m1
mapping.11112 = m1
mapping.11113 = m1
```

Notas:

- *En la configuración del multiplexor de cada uno de los nodos y clientes, se deben de especificar los mismos puertos que utiliza el servidor. Estos números de puertos se llegan a enviar al servidor para que este sepa cómo volver a enrutar el tráfico.*
- *Si se ejecuta en modo local, el host especificado en la configuración de los clientes y los nodos, se debe establecer en localhost o 127.0.0.1, ya que se conectan al multiplexor local.*

Otro caso de uso es cuando se tiene más de un servidor, pero también un cliente o un nodo en la misma máquina. En este caso, el nodo o el cliente no necesitan un multiplexor, ya que se conectará al servidor local JPPF. Esto se ilustra en esta figura:

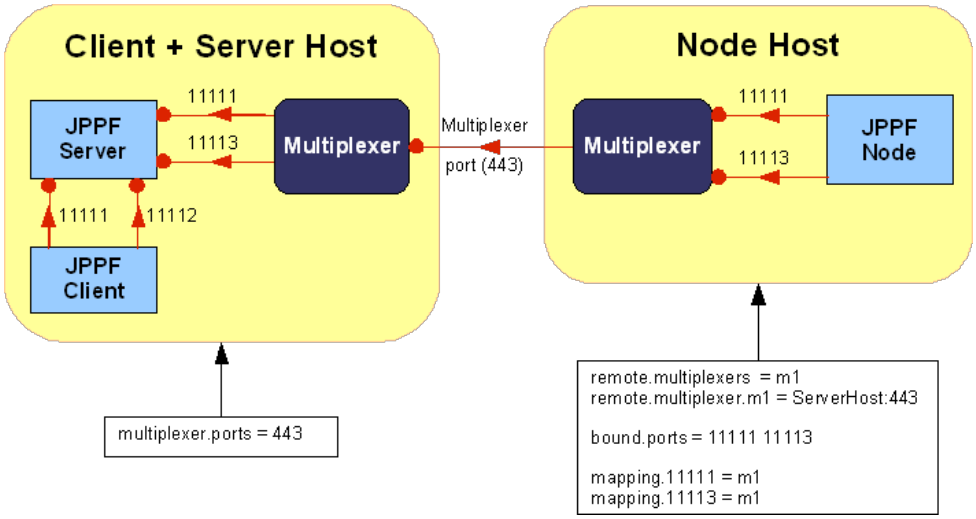


Figura 5.7: Multiplexador en una arquitectura de red extendida.

CAPÍTULO

6 Gestión del proyecto

En este capítulo se analizan las desviaciones que se han producido en cuanto a coste y trabajo respecto a la planificación inicial del proyecto.

6.1. Retrasos

Los retrasos en el desarrollo del proyecto se han producido principalmente por encontrar dificultades al programar, debido al desconocimiento por parte del desarrollador de las principales herramientas que se utilizan en la implementación de la aplicación. Los retrasos se concentran en la implementación de la interfaz gráfica, en la implementación del sistema de fuerza bruta, así cómo, en los sistemas de gestión de archivos que dispone la aplicación. Por otra parte en general ha costado menos tiempo del calculado el diseño de los diferentes sistemas del proyecto: fuerza bruta, seguridad, etc.

6.2. Horas planificadas vs Horas reales

En la tabla 6.1 se puede ver la variación existente entre las horas estimadas para cada una de las tareas con las horas reales.

Nombre de tarea	Trabajo estimado	Trabajo real	Desviación
Preparación del proyecto	**16 horas**	**13 horas**	**-3 horas**
Objetivos del proyecto	8 horas	8 horas	0 horas
Instalación de herramientas	8 horas	5 horas	-3 horas
Formación	**81 horas**	**81 horas**	**0 horas**
Formación de JPPF	30 horas	30 horas	0 horas
Formación de Hazelcast	9 horas	9 horas	0 horas
Formación MySQL	4 horas	4 horas	0 horas
Formación de Jasypt	7 horas	7 horas	0 horas
Formación de Java Swing	16 horas	16 horas	0 horas
Formación de Apache commons	4 horas	4 horas	0 horas
Formación de Látex	11 horas	11 horas	0 horas
Análisis de los requisitos del proyecto	10 horas	10 horas	0 horas
Diseño de la arquitectura general de la aplicación	15 horas	13 horas	-2 horas
Diseño de la base de datos	3 horas	3 horas	0 horas
Implementación de la base de datos	3 horas	5 horas	2 horas
Interfaz modo consola	**20 horas**	**28 horas**	8 horas
Diseño de la interfaz	4 horas	3 horas	-1 horas
Implementación de la interfaz	16 horas	25 horas	9 horas
Interfaz gráfica	**107 horas**	**143 horas**	**36 horas**
Diseño de la interfaz gráfica	4 horas	3 horas	-1 horas
Implementación de la interfaz gráfica	103 horas	140 horas	37 horas
Implementación del controlador	9 horas	12 horas	3 horas
Sistema de fuerza bruta	**263 horas**	**328 horas**	**65 horas**
Diseño del sistema de fuerza bruta	41 horas	30 horas	-11 horas
Implementación del sistema de fuerza bruta	162 horas	221 horas	59 horas
Implementación de la conectividad	60 horas	77 horas	17 horas
Pruebas de conectividad	12 horas	12 horas	0 horas
Sistemas de seguridad	**144 horas**	**161 horas**	**17 horas**
Diseño de los sistemas de seguridad	40 horas	29 horas	-11 horas
Implementación de los sistemas de seguridad	84 horas	112 horas	28 horas
Pruebas	20 horas	20 horas	0 horas
Funcionalidades extra	**54 horas**	**96 horas**	**42 horas**
Implementación de los sistemas de gestion de archivos	38 horas	80 horas	42 horas
Implementación del ataque por diccionario	5 horas	5 horas	0 horas
Implementación de los sistemas de gestion de BBDD	7 horas	7 horas	0 horas
Integración de JPPF Monitor	4 horas	4 horas	0 horas
Pruebas finales y solución de errores	16 horas	16 horas	0 horas
Memoria del proyecto	**113 horas**	**113 horas**	**0 horas**
Manual ténico	70 horas	70 horas	0 horas
Manual de usuario	17 horas	17 horas	0 horas
Elaboración de la presentación	16 horas	16 horas	0 horas
Prepararción de la defensa	10 horas	10 horas	0 horas
Total	**888 horas**	**1.056 horas**	**168 horas**

Tabla 6.1: Horas planificadas vs horas reales

Horas planificadas: 888 horas **Horas reales: 1.056 horas** **Desviación: +168 horas**

6.3. Costes planificados vs Costes reales

Debido al incremento de horas en la realización del proyecto en comparación con las horas planificas, se ha optado por alargar la duración del proyecto en vez de realizar horas extras, con el único fin de abaratar costes.

Concepto	Coste unitario	Tiempo de amortización	Coste unitario de amortización	Timpo de uso	Importe
PC Sobremesa 1	478,90 €	4800	0,09977 €	50 h	4,99 €
PC Sobremesa 2	27,00 €	4800	0,00563 €	50 h	0,28 €
Portátil	1502,99 €	4800	0,31312 €	793 h	248,31 €
Netbook	249,00 €	4800	0,05188 €	50 h	2,59 €
Router Wifi	15,00 €	4800	0,00313 €	48 h	0,15 €
Switch	11,86 €	4800	0,00247 €	48 h	0,12 €
MS Project 2010	775,00 €	4800	0,16146 €	15 h	2,42 €
MS Visio 2010	330,00 €	4800	0,06875 €	6 h	0,41 €
Antivirus Nod32	39.95 €	4800	0,00832 €	843 h	5,53 €
Apache Ant	0,00 €	4800	0,00000 €	378 h	0,00 €
NetBeans 6.9.1	0,00 €	4800	0,00000 €	665 h	0,00 €
MySQL Sever 5.1	0,00 €	4800	0,00000 €	498 h	0,00 €
MySQL GUI Tools 1.2.17	0,00 €	4800	0,00000 €	39 h	0,00 €
IPTools 1.98	0,00 €	4800	0,00000 €	8 h	0,00 €
TeXnicCenter 1.0	0,00 €	4800	0,00000 €	47 h	0,00 €
Ghostscript 8.71	0,00 €	4800	0,00000 €	12 h	0,00 €
GWView 4.9	0,00 €	4800	0,00000 €	12 h	0,00 €
MiKTeX	0,00 €	4800	0,00000 €	47 h	0,00 €
Gimp 2.6	0,00 €	4800	0,00000 €	16 h	0,00 €
Biblioteca Swing	0,00 €	4800	0,00000 €	124 h	0,00 €
Biblioteca Jasypt	0,00 €	4800	0,00000 €	26 h	0,00 €
JPPF Framework	0,00 €	4800	0,00000 €	189 h	0,00 €
Hazelcast Framework	0,00 €	4800	0,00000 €	57 h	0,00 €
Total					**264,80 €**

Tabla 6.2: Amortizaciones reales.

Concepto	Trabajo	Coste	Importe
Desarrollador	1.056 horas	5,30 €/hora	5.596,80 €
Total			**5.596,80 €**

Tabla 6.3: Coste real de los recursos de trabajo.

Los costes de los recursos materiales se mantienen igual que en la planificación ya que su coste se calcula por uso y no ha sido necesario modificar la asignación de recursos de las tareas.

Coste real de recursos materiales: 57,90 €

Concepto	Planificado	Real	Desviación
Recursos de trabajo	4.706,40 €	5.596,80 €	890,40 €
Recursos materiales	57,90 €	57,90 €	0,00 €
Amortizaciones	214,38 €	264,80 €	50,42 €
Total	**4.978,68 €**	**5.919,50 €**	**940,82 €**
Gastos Generales (10 %)	497,86 €	591,95 €	94,09 €
Beneficio (15 %)	746,81 €	887,93 €	141,12 €
Subtotal	**6.223,35 €**	**7.399,38 €**	**1.176,03 €**
IVA (18 %)	1.120,21 €	1331,89	211,68 €
Total	7.343,56 €	8.731,27 €	1.387,71 €

Tabla 6.4: Costes planificados vs costes reales.

Finalmente como muestra la tabla 6.4 los retrasos en el desarrollo del proyecto respecto al tiempo planificado suponen un aumento del coste final de 1.387,71 €, resultando en un coste total de 8.731,27 €.

Parte IV

Manual de usuario

CAPÍTULO

7 Manual de usuario

Este manual está dirigido a desarrolladores, ingenieros de software, arquitectos y cualquier otra persona que desee conocer, aprender o profundizar sus conocimientos en ataques de fuerza bruta distribuidos o ataques por diccionario. Por ello este manual trata de explicar cómo utilizar la aplicación DJ HACK en un entorno gráfico, como logearse en ella, configurar la base de datos, realizar un ataque, etc.

7.1. Prerrequisitos

DJ HACK funciona en cualquier sistema que soporte Java. No se requiere sistema operativo, puede ser instalado en todos los sabores de *Unix*, *Linux*, *Windows*, *Mac OS* y otros sistemas, como *zOS* u otros sistemas *mainframe*.

DJ HACK requiere de las siguientes herramientas instaladas en su máquina:

- Java Standard Edition versión 1.5 o posterior, con la variable de entorno JAVA_HOME apuntando a la carpeta raíz de instalación de Java.
- Apache Ant, la versión 1.7.0 o posterior, con la variable de entorno ANT_HOME apuntando a la carpeta raíz de instalación de Ant.
- Las inscripciones en el PATH del sistema por defecto para JAVA_HOME/bin y ANT_HOME/bin.
- MySQL (Server) 5.1

7.2. Donde descargarlo

DJ HACK se puede descargar desde la forja: `http://sourceforge.net/projects/djhack`

Se ha tratado de dar al fichero que contiene la aplicación un nombre que tenga sentido. El formato es *DJHACK-x.y.z.zip*, donde:

- *x* es el número de versión principal.
- *y* es el número de versión secundaria.
- *z* es el número de lanzamiento del parche - no aparecerá si no se ha establecido ningún parche(es decir, si es igual a 0.)

7.3. Instalación

Cada descarga DJ HACK está en formato *Zip*. Para instalarlo, simplemente descomprímalo en un directorio de su elección.

Cuando lo haya descomprimido, el contenido estará bajo un directorio llamado DJHACK-x.y.z

7.3.1. Cargar de la copia de la base de datos

DJ HACK funciona bajo una base de datos *MYSQL*, para poder utilizar la aplicación es necesario cargar en la base de datos el esquema de la aplicación. Para ello, acceda al *path* de la aplicación, donde se encuentra un fichero llamado *DJHACKDB-x.y.z.sql*. Este es el archivo que se debe cargar en la base de datos.

En *mySQL* existen varios métodos para la realización de un backup, a través de consola o mediante la interfaz gráfica. A continuación se explica la manera de restaurar el backup a través de los dos métodos.

7.3.1.1. Desde la línea de comandos:

Para hacerlo desde la línea de comandos o shell es tan simple como ejecutar el comando siguiente.

```
mysql –user=root –password=miServer djhack
<directorio_ubicación_archivo/DJHACKBD-x.y.z.sql
```

7.3.1.2. Desde el entorno gráfico

Estando en el entorno de trabajo de *MySQL Administrador* en el panel izquierdo deberá elegir la opción *Restore*.

En la ventana que aparece a continuación pulse el botón *Open backup file*, y aparecerá un cuadro de dialogo, como el de la figura 7.1 donde debe abrir el archivo de backup DJHACKBD-x.y.z.sql que contiene la copia de la base de datos se debe restaurar.

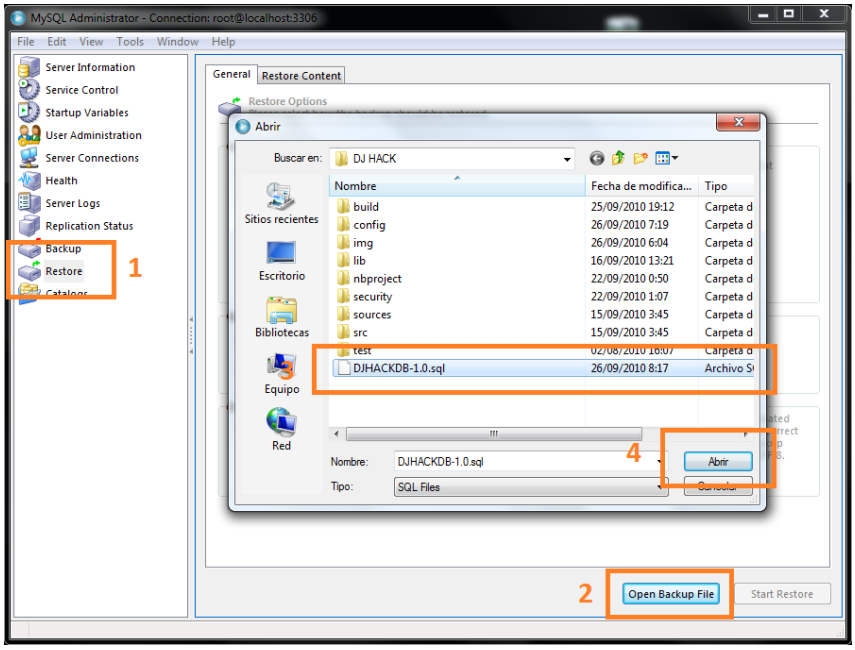


Figura 7.1: Carga del backup en la base de datos I.

Una vez seleccionado el archivo (DJHACKBD-x.y.z.sql), pulse el botón *Start restore*.

Si todo se ejecutó de manera correcta verá una ventana como la que aparece en las figuras 7.2. A continuación y pulse *Close*.

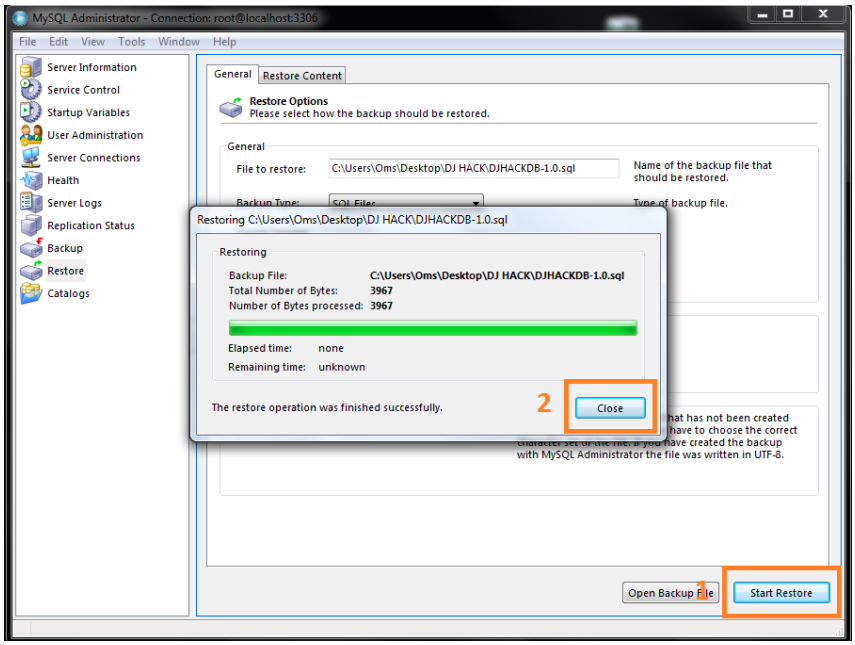


Figura 7.2: Carga del backup en la base de datos II.

7.4. Ejecución de los módulos independientes del clúster

La distribución DJ HACK incluye una serie de módulos independientes o componentes, que se pueden ejecutar de forma independiente en máquinas distintas.

Estos módulos son los siguientes:

- Aplicación: Esta es la aplicación DJHACK, el archivo es DJHACK-x.y.z.zip
- Driver: Es el componente de servidor. Para poder utilizarlo es necesario generarlo a través de la aplicación. Véase la sección 7.5.6
- Nodo: Es el componente del nodo. Para poder utilizarlo es necesario generarlo a través de la aplicación. Véase la sección 7.5.6

Estos módulos están diseñados para ejecutarse desde un *script* de *Ant*. El *script* se llama *build.xml* y siempre tiene un *target* predeterminado llamado *run*. Para ejecutar cualquiera de estos módulos, simplemente escriba "*ant* o *ant run* en el símbolo del sistema o consola de shell.

7.5. Ejecutando DJ HACK

En esta sección se explica el funcionamiento de la aplicación, desde autenticarse hasta realizar ataques de fuerza bruta distribuida.

7.5.1. Autenticación

Nada más arrancar la aplicación, se muestra el cuadro de dialogo encargado de realizar la autenticación al sistema. Debe introducir su nombre de *usuario* y *contraseña* y a continuación pulse en *aceptar*.

Figura 7.3: Autenticación.

NOTA: DJ HACK no permite multiusuario, con lo cual, la primera vez que se inicia el logeo se guardarán los datos introducidos como autenticación Véase la sección de autenticación 7.5.1 de acceso al sistema.

7.5.2. Configuración de la base de datos

Si no se ha realizado previamente la configuración de la base de datos, se le mostrará un cuadro de diálogo *(Figura 7.4)* para realizar dicha configuración.

En ella debe rellenar los campos:

- Usuario: El nombre de usuario para acceder a la base de datos. El usuario debe de estar previamente registrado en la base de datos.
- Contraseña: La contraseña de acceso a la base de datos. Si el usuario no tiene clave de acceso, se debe dejar el campo vacío.
- Dirección IP: Es la dirección IP de la maquina donde está ubicada la base de datos.
- Soporte SSL: Puede cifrar las transmisiones de la base de datos a través de cifrado SSL[1]. Para ello, la base de datos tiene que soportar SSL.

Una vez completado los campos, pulse sobre el botón *aceptar* y la configuración será guardada automáticamente.

Figura 7.4: Configuración del acceso a la base de datos.

[1] Véase: El apartado 5.5

7.5.3. Pantalla principal

En la figura 7.5 se muestran las diferentes herramientas y funcionalidades que dispone DJ HACK. A través de los menús desplegables que se encuentran a la izquierda del programa, podrá acceder a las diferentes secciones que se cargarán en la zona de trabajo ubicada en la parte derecha de la aplicación.

Figura 7.5: Pantalla principal de DJ HACK.

7.5.4. Realizar un ataque

DJHACK dispone de dos tipos de ataque: los de fuerza bruta en entornos distribuidos[2] ,y además, permite la realización de ataques por diccionario en modo local [3]. En los apartados siguientes se explica cómo realizar cada tipo de ataque.

7.5.4.1. Ataque por fuerza bruta

Para realizar un ataque de fuerza bruta, es necesario que los servidores y los nodos estén en funcionamiento *(Véase el apartado de configuración clúster en la sección 7.5.6)*, así como un acceso a la base de datos[4].

Presione el botón *fuerza bruta* en el desplegable de *ataques*.

[2]Véase el apartado 5.1
[3]Véase la sección 5.6.1
[4]Véase como configurar la base de datos 7.5.7.1

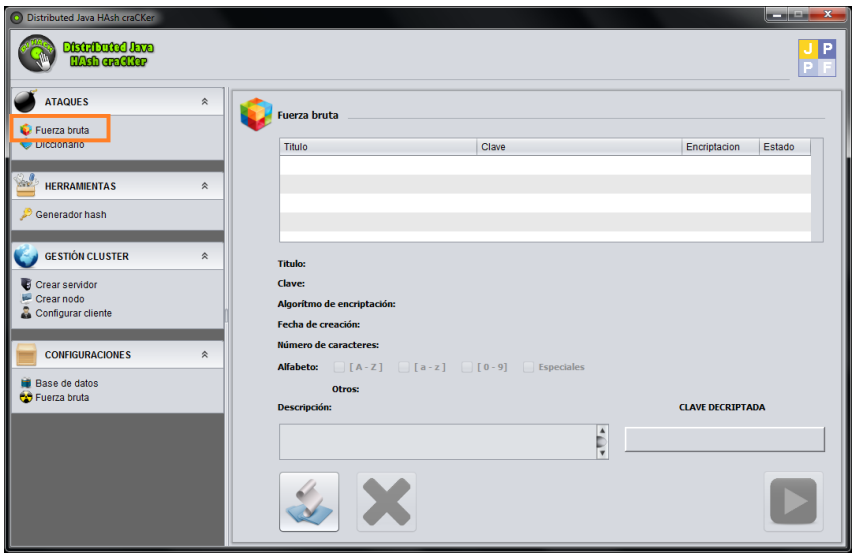


Figura 7.6: Acceso al sistema de fuerza bruta.

Esta herramienta ofrece tres posibilidades: registrar un ataque, eliminar un ataque yrealizar un ataque.

Para registrar un ataque haga clic en el botón de *registrar ataque*. Se abrirá un dialogo con los campos que deberá rellenar para registrar un ataque *(Véase la figura 7.7)*.

A continuación se muestra la explicación de cada uno de los campos que debe rellenar:

- Titulo: El nombre que identifica el ataque.
- Clave: La clave a decriptar en formato hexadecimal.
- Algoritmo de encriptación: El algoritmo a usar para decriptar la clave.
- Descripción: Puede añadir una descripción detallada del ataque.
- Alfabeto: El alfabeto a utilizar en la descripción.
  - [A - Z]: Se incluyen en el alfabeto caracteres en mayúscula de la *A-Z* exceptuando la *ñ* que se considera caracter especial.
  - [a - z]: Se incluyen en el alfabeto caracteres en minúscula de la *a-z* exceptuando la *ñ* que se considera caracter especial.
  - [0 - 9]: Se incluyen en el alfabeto los números ordinales del *0 al 9*.
  - Especiales: Se incluyen en el alfabeto caracteres especiales. Los caracteres considerados como especiales son: `ñ|!@#~\%\$-+:ºª¿?!¡<>()[]{}`
  - Otros: Ud. puede definir exclusivamente el alfabeto que quiera incluir a su ataque, o incluso podría enlazarlos con las otras opciones.

- Número de caracteres: La cantidad posible de caracteres que puede tener la clave real. Se puede especificar a través de comas *(Ejemplo: 5,8)* si desea realizar un ataque más específico, o mediante un guion *(Ejemplo: 2-16)* si desea realizar un ataque más amplio *(Por ejemplo: introducir 2-16, equivaldría a insertar 2, 3, 4, 5, 6, , 16).*

Una vez especificados todos los parámetros requeridos para realizar el ataque, pulse al botón *guardar* y su ataque será registrado en el sistema.

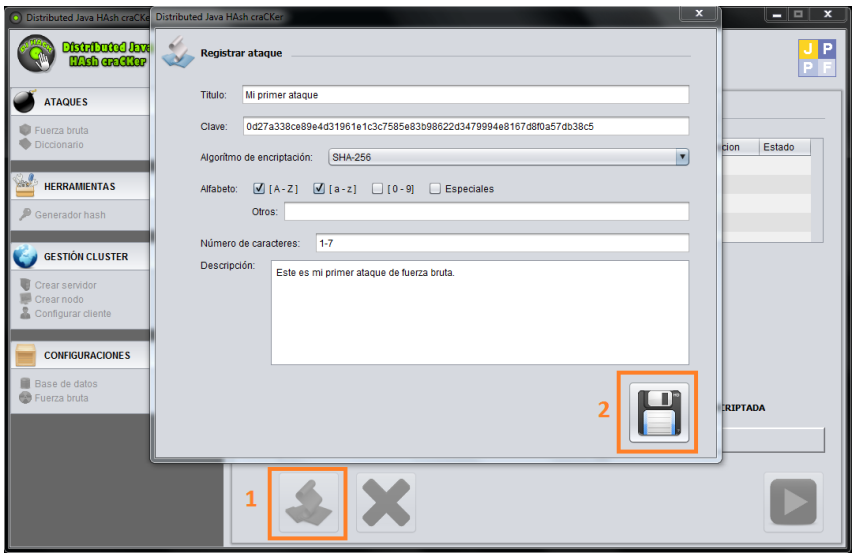


Figura 7.7: Registro de un ataque.

A continuación, volverá automáticamente a la pantalla principal. Como se puede apreciar en la tabla que aparece en la figura7.8, se ha almacenado el nuevo registro.

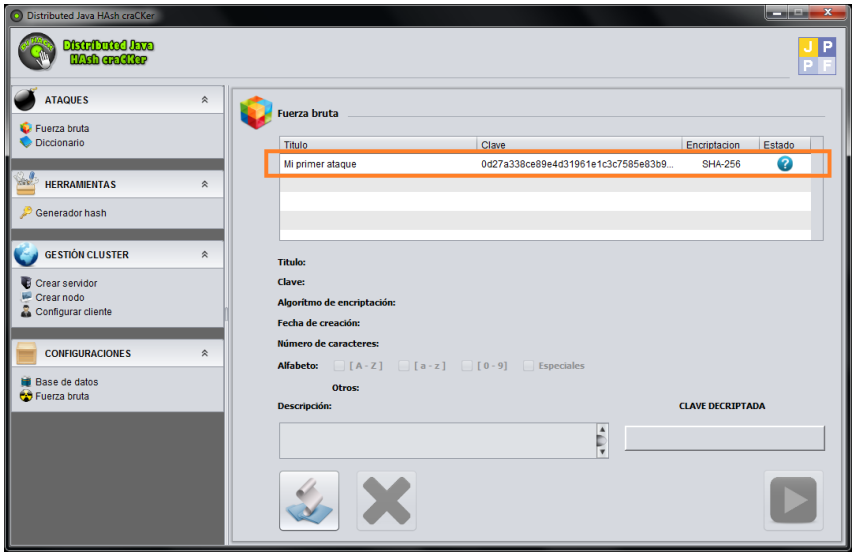


Figura 7.8: Registro almacenado.

Haciendo clic sobre la fila de la tabla donde se encuentra el registro a gestionar, se puede ver, en la parte inferior de la aplicación, las características completas del ataque: título, clave, fecha de creación, descripción, estado, etc.

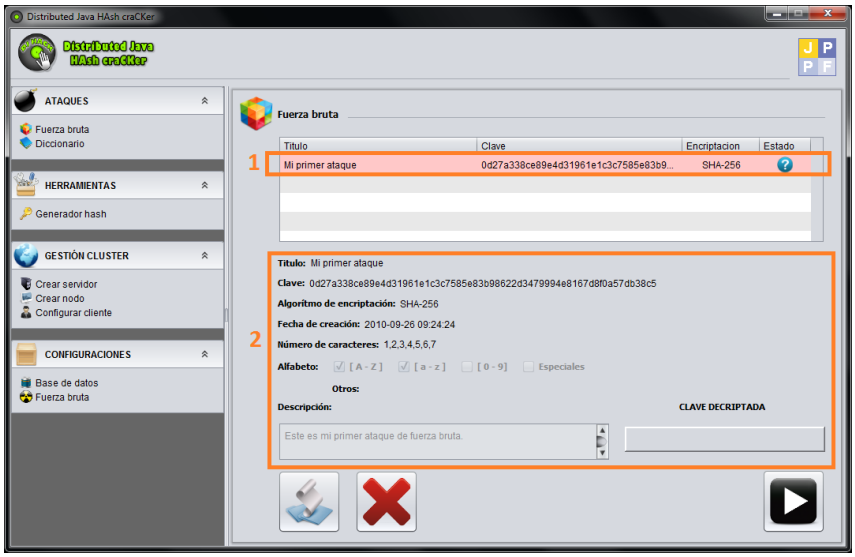


Figura 7.9: Características del registro.

Además, se puede eliminar los registros que no desee tener almacenados, clicando sobre uno de ellos en la tabla y pulsando el botón *eliminar*. En la figura 7.10 se puede ver cómo eliminar un registro.

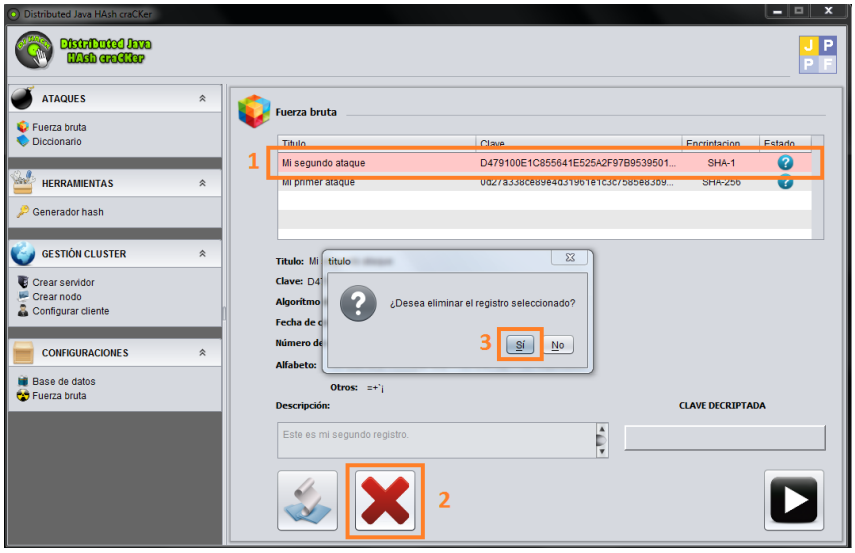


Figura 7.10: Eliminación de un registro.

Una vez seleccionado el registro al que se quiere realizar un ataque, únicamente debe pulsar el botón *atacar*. Seguidamente se mostrara un dialogo determinado que se está realizando el ataque.

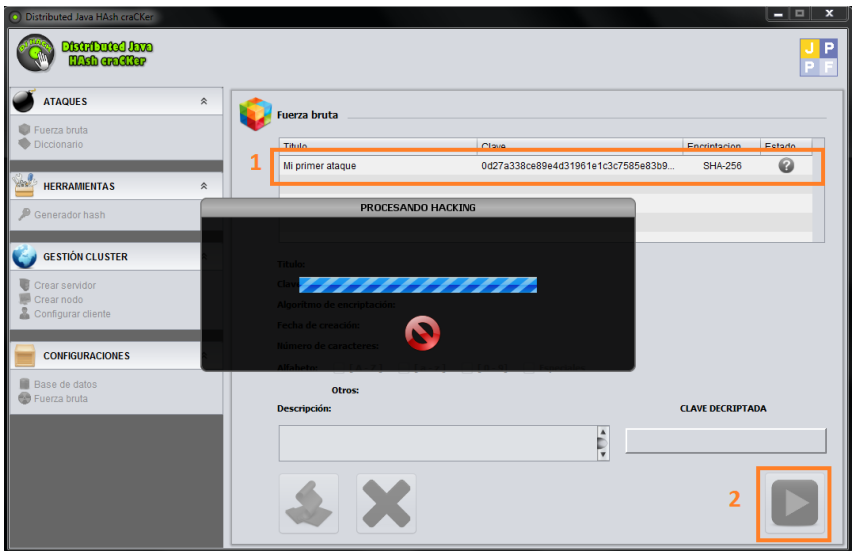


Figura 7.11: Realización de un ataque de fuerza bruta.

El tiempo de realización del ataque a traves de la fuerza bruta, puede demorar mucho tiempo, dependiendo de la seguridad que tenga la clave a decriptar.

No obstante, pulsando al botón *cancelar* del dialogo, puede detener el ataque en el momento que lo precise. Sea paciente, este proceso puede tardar un cierto tiempo. De todos modos, usted puede reanudar el ataque posteriormente cuando lo desee.

Figura 7.12: Cancelación de un ataque de fuerza bruta.

Una vez finalizado el ataque, o de haber sido cancelado antes de acabar, se pueden mostrar tres resultados diferentes en el apartado de *clave decriptada*.

- Clave encontrada: Si se ha finalizado correctamente el ataque y se ha encontrado la contraseña.

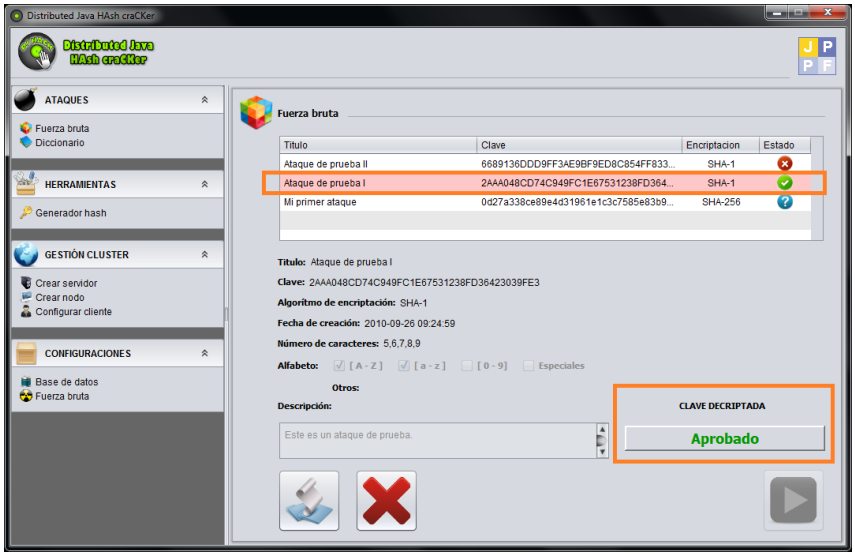


Figura 7.13: Visualización de una clave decriptada.

- Clave no encontrada: Si se ha finalizado correctamente el ataque, pero no se ha encontrado la contraseña con los parámetros definidos.

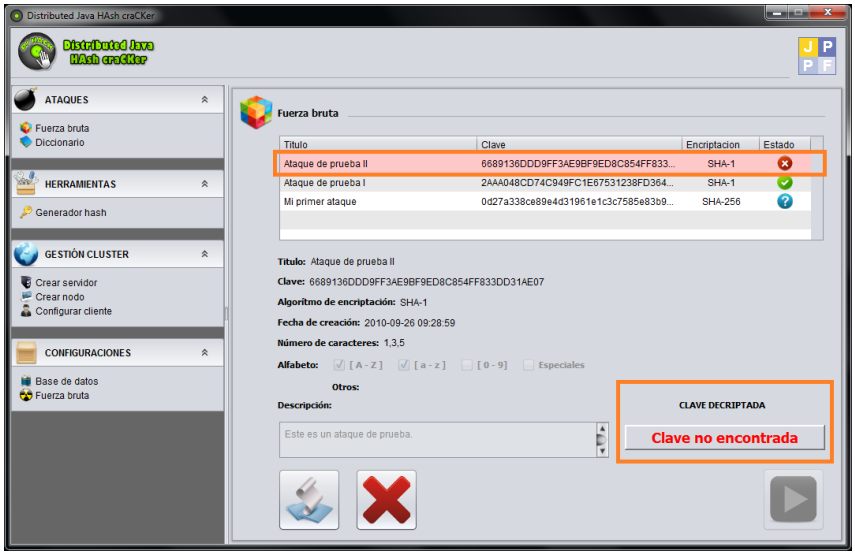


Figura 7.14: Visualización de un ataque de fuerza bruta fallido.

- Sin finalizar: Si se ha cancelado el ataque, el apartado estado de la clave no mostrará ningún mensaje. No obstante, puede usted reanudar el ataque en el momento que lo desee.

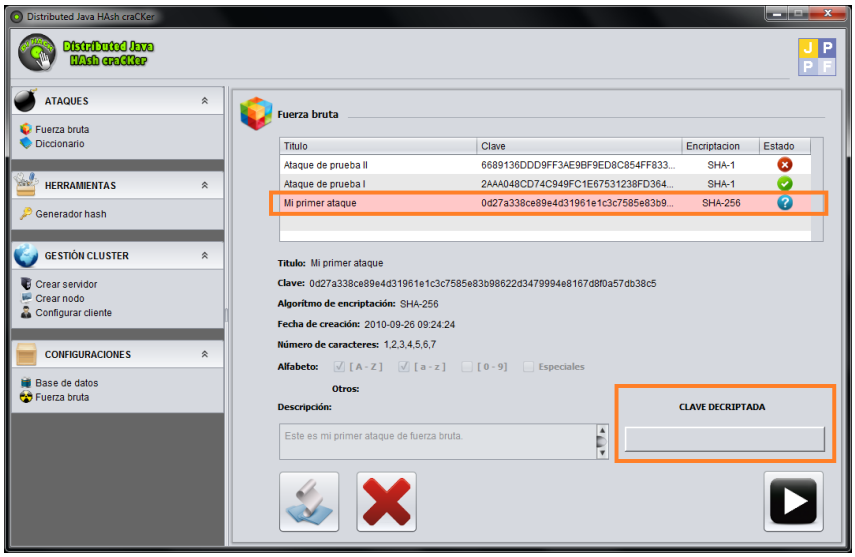


Figura 7.15: Visualización de un ataque sin finalizar.

7.5.4.2. Ataque por diccionario

Para realizar un ataque de diccionario, únicamente es necesario un acceso a la base de datos[5].

Presione el botón de *diccionario* en el desplegable de *ataques*.

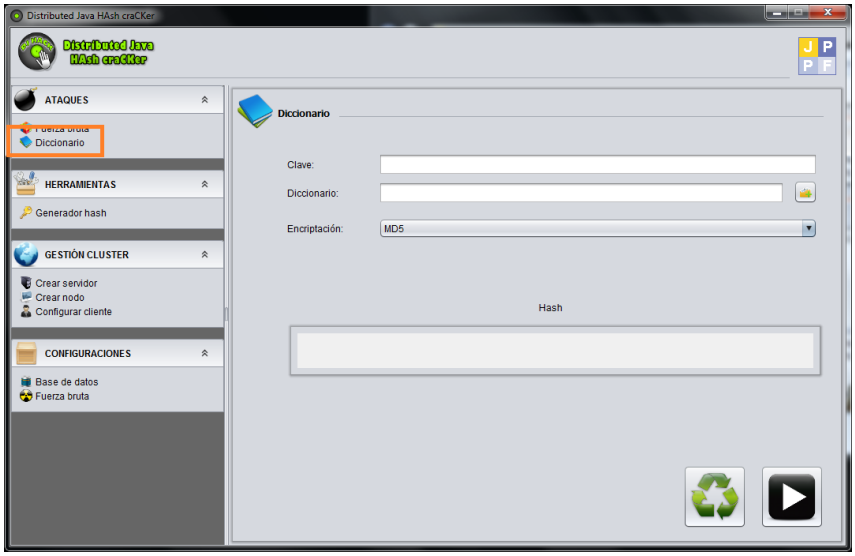


Figura 7.16: Acceso al sistema de diccionario.

A continuación, se le cargará en la pantalla de la derecha el sistema de gestión de ataques de diccionario. Para ello debe importar un diccionario[6] pulsando en el botón de la carpeta, e introducir los datos de la clave a decriptar y el algoritmo a utilizar en la decriptación. *(Figura 7.17)*.

Seguidamente, pulse en el botón de *realizar ataque* y se abrirá un dialogo mostrando[7] que se está realizando el ataque.

Finalmente, una vez finalizado el ataque se mostrará el resultado como ilustra la figura 7.18.

[5] Véase cómo configurar la base de datos 7.5.7.1.

[6] DJ HACK únicamente soporta diccionarios con extensión *.txt*. El contenido del diccionario debe de estar estructurado de la siguiente manera: palabra1 [salto de línea] palabra2 [salto de línea] ...

[7] Los ataques por diccionario son tan rápidos, que a veces no es posible contemplar el dialogo de información.

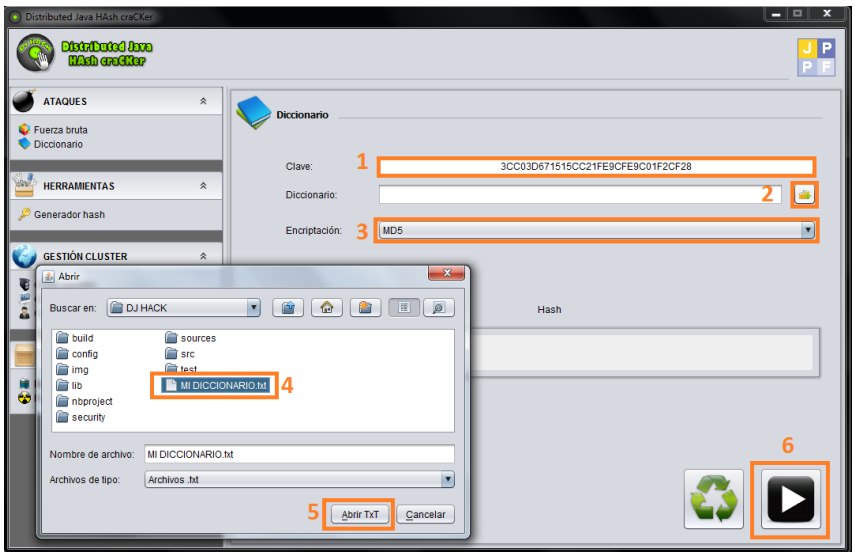


Figura 7.17: Realización de un ataque de diccionario.

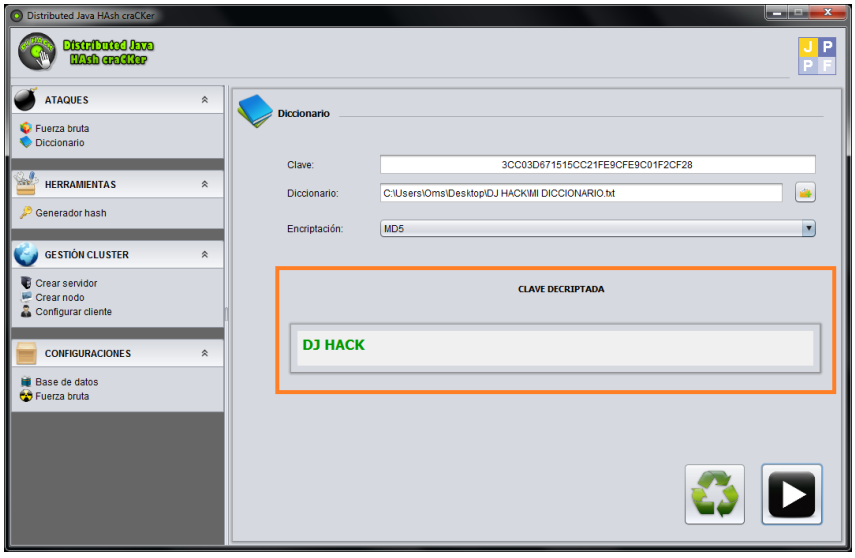


Figura 7.18: Visualización de los resultados obtenidos.

7.5.5. Herramientas. Generador hash

Pulsando sobre el botón *generador hash*, en la pestaña *herramientas*, usted puede generar sus propias claves hash.

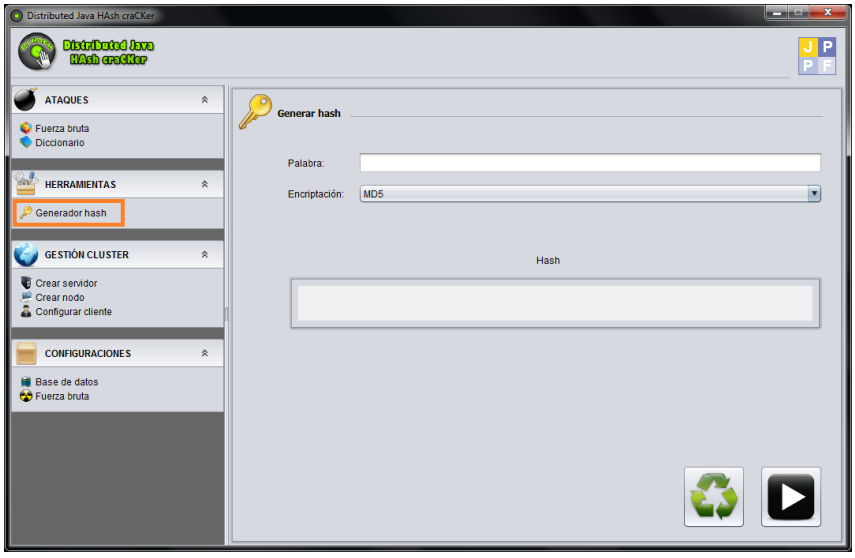


Figura 7.19: Acceso al generador hash.

Para ello debe de indicar la palabra a encriptar y el algoritmo a utilizar en la encriptación, a continuación pulse sobre el botón *aceptar* y en el capo *clave hash* aparecerá la clave encriptada.

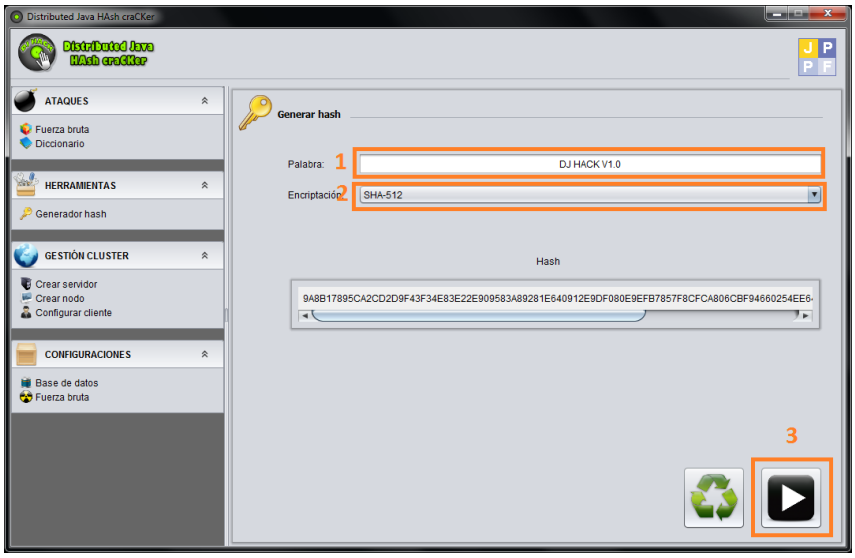


Figura 7.20: Creación de un palabra hash.

7.5.6. Configuration clúster

DJ HACK proporciona sistemas automatizados para la generación y gestión de los componentes (cliente, servidores y nodos) del clúster.

7.5.6.1. Crear servidor/nodo

Los pasos a seguir para crear un servidor o un nodo son iguales, por lo que en este manual solo se detalla la creación de los servidores.

Para crear un servidor, pulse en el botón de *crear servidor* en la pestaña *configuración clúster*. En el panel que se muestra a la izquierda puede indicar el directorio donde quiere crear los servidores, la cantidad de servidores a crear y la configuración que dispondrán.

Una vez rellenados los campos pulse el botón de *ejecutar*. Y si desea restaurar la configuración por defecto, pulse sobre el botón *restaurar*.

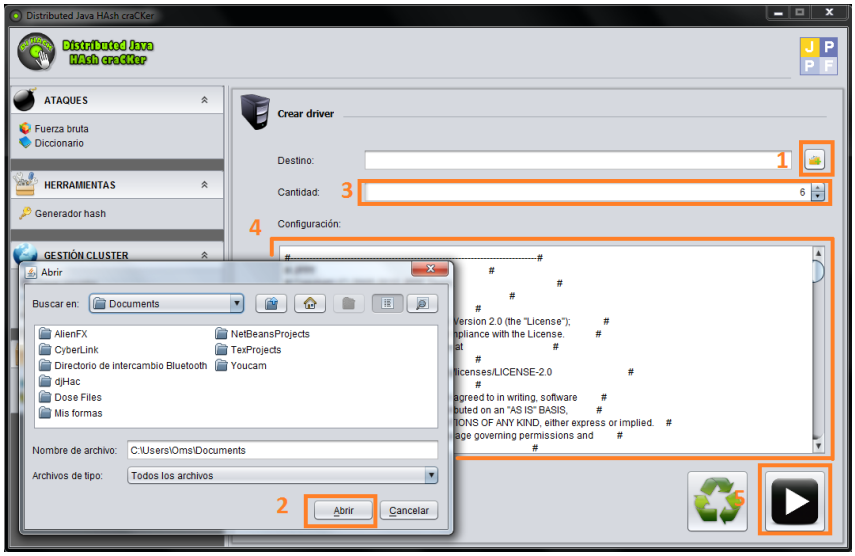


Figura 7.21: Creación de servidores II.

7.5.6.2. Configurar cliente

Puede modificar la configuración del cliente accediendo a la sección de *configurar cliente*.

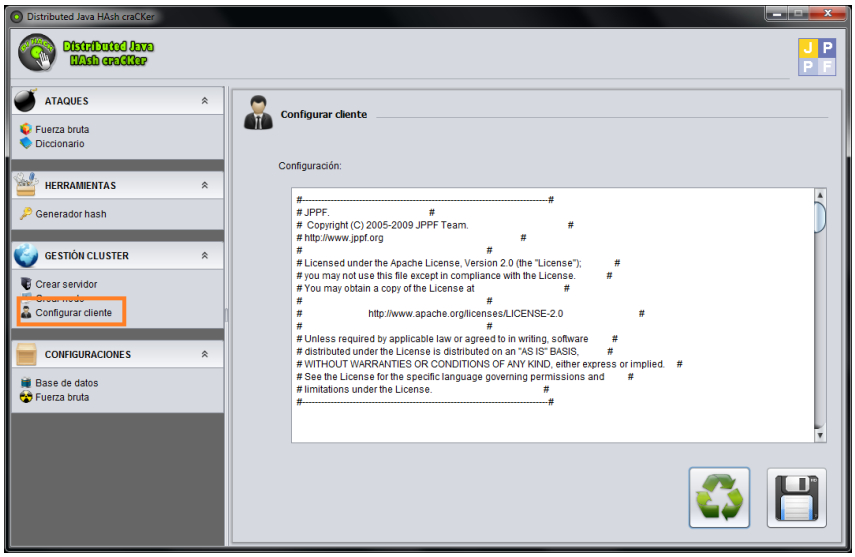


Figura 7.22: Configuración del cliente I.

Una vez modificada la configuración usted puede establecerla pulsando sobre el botón *establecer*, o restaurar la configuración por defecto pulsando sobre el botón *restaurar*.

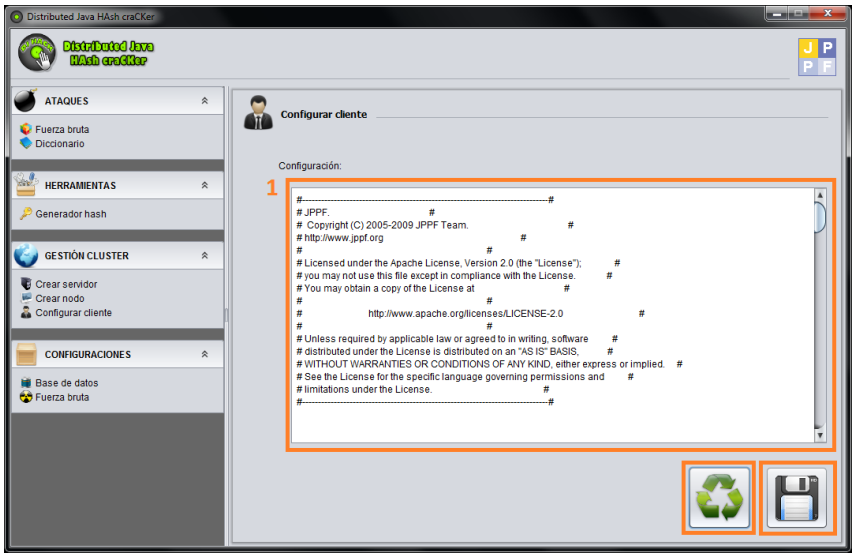


Figura 7.23: Configuración del cliente II.

7.5.7. Configuraciones

DJ HACK proporciona dos tipos de configuraciones, una para la base de datos y otra para los ataques de fuerza bruta.

7.5.7.1. Configuración base de datos

La configuración de la base de datos puede modificarse en cualquier momento. Para acceder a ella, haga clic en la pestaña *configuraciones* y pulse sobre el botón *configuración base de datos*. En el panel de la derecha se mostrará la pantalla encargada de realizar la configuración.

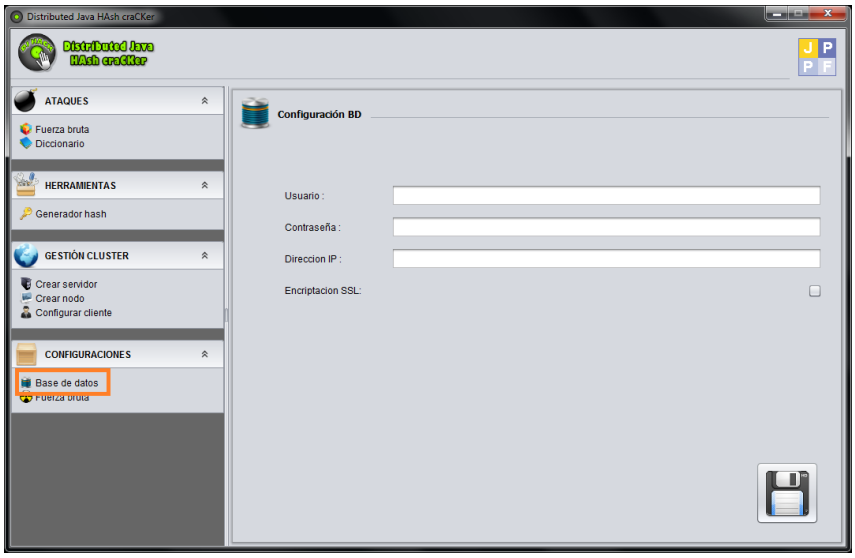


Figura 7.24: Configuración base de datos en la aplicación.

Para realizar la configuración únicamente debe de rellenar los campos al igual que se realiza en el apartado 7.5.2. Y pulsar el botón de *guardar*, como se muestra en la figura **??**.

7.5.7.2. Configuración fuerza bruta

Ud. puede ajustar los parámetros clave que se necesitan en la configuración de la fuerza bruta. Por un lado, la cantidad de palabras auxiliares que generará la computadora desde donde se ejecuta aplicación, y por otro lado el número de palabras que generará cada uno de los procesadores de cada nodo. *Véase más información en el apartado 5.1.3.2.*

Para poder acceder a este sistema, haga clic en el boton *configuración fuerza bruta*.

Para ajustar estos parámetros *aumente* o *disminuya* la cantidad de palabras que utiliza cada uno de los apartados, y posteriormente pulse sobre el botón *aceptar*. En cambio si desea volver a la configuración por defecto, pulse sobre el botón, *restaurar*.

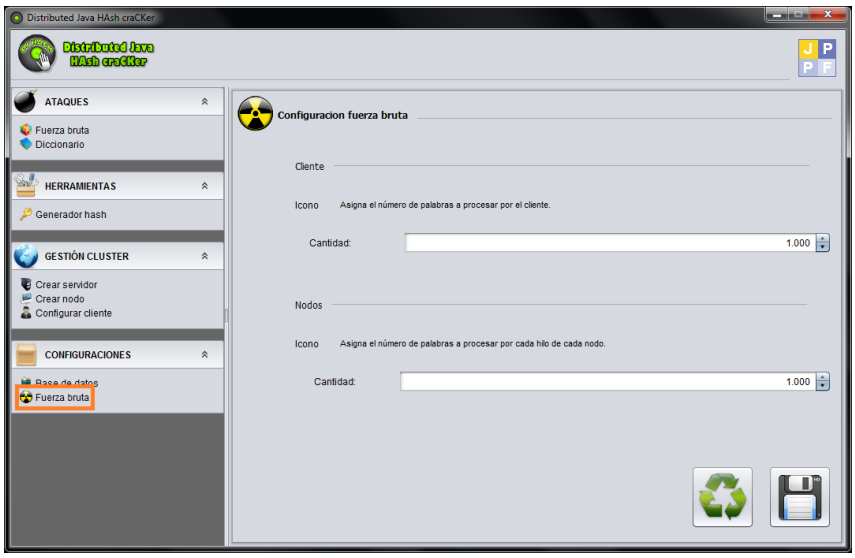


Figura 7.25: Configuración de la fuerza bruta I.

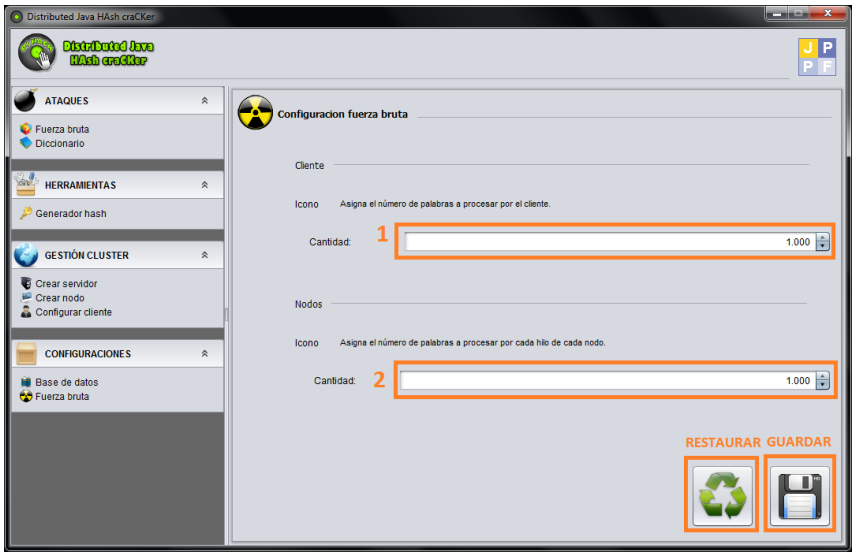


Figura 7.26: Configuración de la fuerza bruta II.

7.6. Implantación del sistema de encriptación

DJ HACK provee de un sistema encargado de encriptar las comunicaciones entre la aplicación, los servidores y los nodos mediante una codificación DES[8]

En primer lugar usted necesita construir el archivo *jar* encargado de realizar la encriptacion. Para hacer esto siga los siguientes pasos:

1. Acceda a la carpeta *add-ons/DataEncryption* que se encuentra ubicada en el directorio raíz de la aplicación.

[8]Para más información sobre el algoritmo DES acceda a la sección 3.3.4.4 de 56 bits.

2. Abra el archivo *build.xml* con un editor de texto, y modifique la propiedad *password*, asignándole como valor la contraseña que se utilizará para el almacén de llaves (*keystore*).

3. Acceda mediante la línea de comandos a la carpeta *add-ons/DataEncryption* y genere el archivo *DataEncription.jar* escribiendo *ant jar*.

El siguiente paso es desplegar el archivo jar en todos los componentes de la red, incluidos los servidores, nodos y la aplicación. Para conectarlo al componente JPPF:

1. Agregue el archivo *jar* a la ruta de clase de cada componente: En el caso de un servidor o nodo, es simplemente una cuestión de ubicarlo en la carpeta *lib* de ubicada en la carpeta raíz de cada componente. Para la aplicación elimine el archivo *DataEncription.jar* que se encuentra en la carpeta *lib* de la aplicación e inserte el *jar* que usted ha generado.

2. Edite el archivo de configuración JPPF de cada componente y descomente la siguiente propiedad:

```
jppf.data.transform.class =
org.jppf.example.dataencryption.SecureKeyCipherTransform
```

Una vez hecho esto, puede reiniciar los servidores, nodos y los clientes, y todos los datos enviados a través de la red serán automáticamente cifrados.

7.7. Implantación del sistema de multiplexación

Esta aplicación dispone de un sistema de multiplexación[9] de puertos para entornos con cortafuegos restrictivos.

Para implantar la multiplexación acceda a la carpeta *add-ons/multiplexer* ubicada en el directorio raíz de la aplicación y siga los siguientes pasos:

- Copie la carpeta en todas las maquinas del grid donde están ubicados los nodos, servidores y la aplicación

- Para el equipo del servidor:

  - Edite el archivo *config/multiplexor-server.properties* y establezca los valores adecuados para su entorno.

[9] Véase más información en sobre el uso de la multiplexación de puertos en la sección 5.6.6.1

- Desde una shell o símbolo del sistema, ejecute en desde la carpeta *multiplexer ant run.multiplexer.server*.

- Para la aplicación o nodos:

  - Edite el archivo *config/wmultiplexer.properties* y establecer los valores adecuados para su entorno.
  - Desde una shell o símbolo del sistema, ejecute en desde la carpeta *multiplexer ant run.multiplexer*

Nota: Puede ver varios ejemplos de configuración según la topología de red, en el apartado 5.6.6.1

Parte V

Conclusiones y trabajo futuro

CAPÍTULO

8 Conclusiones y líneas futuras

Este capítulo muestra la conclusión a la que se ha llegado sobre la aplicación desarrollada, y comenta las lineas de trabajo futuro que puede llegar a tener.

8.1. Conclusiones

Teniendo en cuenta el fuerte avance en las tecnologías de la encriptación y la informática, resulta sorprendente que no existan herramientas tan completas y sencillas como ésta para realizar auditorías criptográficas distribuidas, y poder comprobar la seguridad de las contraseñas.

DJ HACK es una aplicación potente y versátil, donde cualquier persona la pueda utilizar sin tener conocimientos avanzados sobre la criptografía, ni el cómputo distribuido. Por ello, esta aplicación se ha diseñado especialmente orientada al usuario, con la finalidad de automa-

tizar cada uno de los procesos que se necesitan para la computación en grid y proporcionando mecanismos sencillos para su implantación en una red.

La finalidad este programa se ha basado en obtener una herramienta útil para realizar auditorías criptográficas, y que además tiene mucho margen para crecer. Algunas ideas que se plantearon al proponer este proyecto se han ido desechando por falta de tiempo o por centrarse en otros aspectos que se han considerado más importantes. Y otras muchas han ido surgiendo a medida que se desarrollaba el proyecto.

En cuanto a las tecnologías usadas en esta aplicación, se puede decir, que han sido realmente variadas y muchas de ellas absolutamente desconocidas para el desarrollador. Además del proceso de desarrollo de la propia aplicación, el desarrollador se visto obligados a buscar información y hacer infinidad pruebas para entender el funcionamiento de estas tecnologías. Es por esto que una parte importante del esfuerzo volcado en este proyecto ha consistido en profundizar en los conocimientos de JPPF, Hazelcast, criptografía, protocolos de seguridad, etc.

8.2. Líneas futuras

DJ HACK es un proyecto que tiene un gran potencial para crecer y probablemente también muchos aspectos que pueden ser mejorados a continuación describiremos algunas funcionalidades que pueden implementarse en un futuro:

- Aumentar la seguridad de las transmisiones entre cliente, servidores y nodos. Proporcionando técnicas criptográficas más avanzadas y seguras que el algoritmo *DES*. Como por ejemplo, cifrar todas las transmisiones a través de comunicaciones seguras *SSL*.
- Mejorar el sistema de ataque por diccionario, proporcionando la posibilidad de introducir diferentes tipos de ficheros y que el sistema los reconozca sin ningún problema.
- Acoplar a la aplicación un programa, que sea capaz de realizar ataques mediante el uso de las *RainbowTables* (*tablas arcoíris*)
- Añadir sistemas, como *JCUDA*, para paralelizar en gran medida el proceso de ruptura de claves, aprovechando la potencia de las nuevas *GPUs*

Parte VI

Apéndices y bibliografía

APÉNDICE

A Pliego de condiciones

A.1. Condiciones del cliente

Este apartado hace alusión a las condiciones mínimas requeridas para hacer uso de la aplicación. Las características mínimas que se requieren en el ordenador para hacer uso del servicio son las siguientes:

Para Microsoft Windows o Linux:

- Un procesador Intel Pentium IV o equivalente a 800Mhz o más rápido.
- 1GB de memoria RAM.
- Monitor de 2 colores (para la versión server) con una capacidad para mostrar una resolución 800x600. Pero lo recomendado seria, un monitor de 256 colores con capacidad de mostrar una resolución de 1024x768 píxeles (para la versión gráfica).

Para Macintosh:

- Un Power Macintosh con Mac OS 8.6 ó 9.x.
- 1GB de memoria RAM.
- Monitor de 256 colores con capacidad para mostrar una resolución de 800x600 píxeles, aunque se recomienda una resolución de 1024x768 píxeles.

Ratón:Es recomendable su utilización para una mayor comodidad del usuario, aunque su uso no es obligatorio, especialmente en la versión server.

Monitor:La aplicación ha sido realizada en una resolución de 1024x768 píxeles, luego esta será la resolución recomendada para una mejor visualización de la aplicación, aunque no es obligatorio.

A.2. Condiciones del servidor y otros elementos hardware

Las características recomendadas para el ordenador que ejerza de servidor y para un correcto funcionamiento son los siguientes:

- Un mínimo de un servidor con dos o más procesadores cada uno.
- Los servidores deben disponer de tecnología de 64 bits completa.
- Los servidores deben soportar todos los estándares de Internet como TCP/IP, DNS, NTP, DHCP entre otros.
- De 64 bits: 64 bits de espacio de direcciones virtuales, y mínimo 50 bits de espacio de direcciones físicas.
- Microprocesadores de 1.3 GHz o superior.
- Memoria caché L2/L3 de 3MB o superior por procesador.
- 2GB de memoria RAM.
- Una tarjeta de red 10/100/1000.
- Disponer de esquemas de control de acceso (usuarios, procesos).
- Un switch de red con rendimiento punto a punto de al menos 200 MBs por puerto.

A.3. Condiciones del software

Considerando los objetivos del cliente, la única necesidad de software necesaria en todos los dispositivos que se utilizarán para poder realizar ataques distribuidos son:

- JDK 6.
- Apache Ant
- MySQL Server 5.1[1]

A.4. Condiciones generales

La presente aplicación ha sido diseñada como una propuesta del departamento de Lenguajes y Sistemas Informáticos de la Escuela Universitaria de Ingeniería de Vitoria-Gasteiz de la UPV-EHU. Con ella se ha pretendido diseñar un sistema útil y amigable para realizar auditorías criptográficas en entornos distribuidos, además de permitir de forma automatizada la creación de cada uno de los módulos que utiliza la aplicación, así como monitorizar en tiempo real el funcionamiento de estos.

Se busca facilitar el trabajo del usuario, ya que tendrán a su disposición una aplicación en la que se tendrán centralizadas todas las herramientas imprescindibles para la implantación de la aplicación en la red.

Estimando que los datos aportados en el presente pliego de condiciones son suficientes, se somete el proyecto a su aprobación por el departamento de LSI.

Vitoria-Gasteiz, 23 de Septiembre de 2010

[1] MySQL Server 5.1 únicamente debe de instalarse en una máquina del servidor.

APÉNDICE

B Código centinela

A continuación se muestra la clase encargada de inicializar todos los nodos cuando se realiza un ataque de fuerza bruta.

```
package packFuerzaBruta;

import com.hazelcast.core.EntryEvent;
import com.hazelcast.core.EntryListener;
import com.hazelcast.core.IMap;
import java.util.concurrent.atomic.AtomicBoolean;
import org.jppf.server.protocol.JPPFTask;
import packInicializacion.packInicializar.packNodo.NodoStartup;
```

```
/**
 * <p>Inicializacion distribuida. Clase encargada de restablecer los
    campos del mapa distribuido de Haselcast cada
 * vez que se realice un hackeo.</p>
 *
 * @author Unai Gómez Velasco
 * @version 1.0
 */
public class IncializacionDistribuida extends JPPFTask implements
   EntryListener{

    /**
     * Elimina el registro  del mapa distribuido Haselcast en el caso
         de que exista ,que determina si se ha encontrado o no una
        clave,
     *
     */
    @Override
    public void run() {
        System.out.println("Se estan recalculando los nodos");
        //Se recoge la instancia del mapa distribuido.
        IMap<String, AtomicBoolean> mapaDistribuido = NodoStartup.
           getInstance().getMapaDistribuido();
        /*Se anadimos como evento de escucha por todos los nodos,
           este nodo.
        Asi, en el caso de que se produzca una entrada en hazelcast
           todos
        la escucharan.*/
        mapaDistribuido.addEntryListener(this, true);
        /*Si el mapa dispone de algun dato (la condicion de que la
           clave ha sido decriptada)
        significa que se ha cumplido la condicion necesaria y
           suficiente para restablecer los
         campos por defecto.*/
        if (!mapaDistribuido.isEmpty()) {
            mapaDistribuido.remove("CondicionAtomica");
        }
    }

    @Override
    public void entryAdded(EntryEvent ee) {
```

```
    }

    @Override
    public void entryRemoved(EntryEvent ee) {
    }

    @Override
    public void entryUpdated(EntryEvent ee) {
    }

    @Override
    public void entryEvicted(EntryEvent ee) {
    }

}
```

APÉNDICE

C Configuraciónes

En este apéndice se muestran las configuraciones de cada uno de los componentes de clúster.

C.1. Cliente

```
#---------------------------------------------------------------#
# JPPF.
#  Copyright (C) 2005-2009 JPPF Team.
# http://www.jppf.org
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#      http://www.apache.org/licenses/LICENSE-2.0
```

```
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#---------------------------------------------------------------#




#---------------------------------------------------------------#
# List of drivers this client may connect to.
# If auto discovery of the server is enabled, this needs not be
   specified.
#---------------------------------------------------------------#

#jppf.drivers = driver1

#---------------------------------------------------------------#
# Host name, or ip address, of the host the JPPF driver is running on
# If auto discovery of the server is enabled, this needs not be
   specified.
#---------------------------------------------------------------#

#driver1.jppf.server.host = 192.168.0.193

#---------------------------------------------------------------#
# port number for the class server that performs remote class loading
# default value is 11111; uncomment to specify a different value
# If auto discovery of the server is enabled, this needs not be
   specified.
#---------------------------------------------------------------#

#driver1.class.server.port = 11111

#---------------------------------------------------------------#
# port number the clients / applications connect to
# default value is 11112; uncomment to specify a different value
                  # If auto discovery of the server is enabled, this
   needs not be specified.
```

```
#---------------------------------------------------------#

#driver1.app.server.port = 11112

#---------------------------------------------------------#
# JMX management port of the driver
# default value is 11198; uncomment to specify a different value
# If auto discovery of the server is enabled, this needs not be
   specified.
#---------------------------------------------------------#

#jppf.management.port = 11098
#jppf.management.enabled = false

#---------------------------------------------------------#
# Priority given to the driver
# The client is always connected to the available driver(s) with the
   highest
# priority. If multiple drivers have the same priority, they will be
   used as a
# pool and tasks will be evenly distributed among them.
# default value is 0; uncomment to specify a different value
#---------------------------------------------------------#

#driver1.priority = 10
#driver1.jppf.pool.size = 10

#driver2.jppf.server.host = localhost
#driver2.class.server.port = 11121
#driver2.app.server.port = 11122
#driver2.priority = 10

#---------------------------------------------------------#
# Maximum time in milliseconds spent trying to initialize at least
   one
# connection, before releasing control to the main application thread
   .
# default value is 1000 (1 second); uncomment to specify a different
   value
#---------------------------------------------------------#
```

```
#jppf.client.max.init.time = 1000

#---------------------------------------------------------------#
# Automatic recovery: number of seconds before the first reconnection
    attempt.
# default value is 1; uncomment to specify a different value
#---------------------------------------------------------------#

#reconnect.initial.delay = 1

#---------------------------------------------------------------#
# Automatic recovery: time after which the system stops trying to
   reconnect,
# in seconds. A value of zero or less means the system nevers stops
   trying.
# default value is 60; uncomment to specify a different value
#---------------------------------------------------------------#

reconnect.max.time = -1

#---------------------------------------------------------------#
# Automatic recovery: time between two connection attempts, in
   seconds.
# default value is 1; uncomment to specify a different value
#---------------------------------------------------------------#

#reconnect.interval = 1

#---------------------------------------------------------------#
#  Enable local execution of tasks? Default value is false
#---------------------------------------------------------------#

jppf.local.execution.enabled = true

#---------------------------------------------------------------#
# Number of threads to use for loacal execution
# The default value is the number of CPUs available to the JVM
#---------------------------------------------------------------#

#jppf.local.execution.threads = 4
```

```
#-------------------------------------------------------------#
# Enable/Disable automatic discovery of JPPF drivers.
# default value is true; uncomment to specify a different value
#-------------------------------------------------------------#

#jppf.discovery.enabled = true

#-------------------------------------------------------------#
# UDP multicast group to which drivers broadcast their connection
   parameters
# and to which clients and nodes listen. Default value is 230.0.0.1
#-------------------------------------------------------------#

#jppf.discovery.group = 230.0.0.1

#-------------------------------------------------------------#
# UDP multicast port to which drivers broadcast their connection
   parameters
# and to which clients and nodes listen. Default value is 11111
#-------------------------------------------------------------#

#jppf.discovery.port = 11111

#jppf.data.transform.class = org.jppf.example.dataencryption.
   SecureKeyCipherTransform
```

C.2. Servidor / Driver

```
#-------------------------------------------------------------#
# JPPF
# Copyright (C) 2005-2010 JPPF Team.
# http://www.jppf.org
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#      http://www.apache.org/licenses/LICENSE-2.0
# Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied.
```

```
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------#

#-------------------------------------------------------------#
# Host name, or ip address, of the host the JPPF driver is running on
#-------------------------------------------------------------#

jppf.server.host = localhost

#-------------------------------------------------------------#
# port number for the class server that performs remote class loading
# default value is 11111; uncomment to specify a different value
#-------------------------------------------------------------#

#class.server.port = 11111

#-------------------------------------------------------------#
# port number the clients / applications connect to
# default value is 11112; uncomment to specify a different value
#-------------------------------------------------------------#

#app.server.port = 11112

#-------------------------------------------------------------#
# port number the nodes connect to
# default value is 11113; uncomment to specify a different value
#-------------------------------------------------------------#

#node.server.port = 11113

#-------------------------------------------------------------#
# Enabling JMX features
# default value is true; uncomment to specify a different value
#-------------------------------------------------------------#

#jppf.management.enabled = true

#-------------------------------------------------------------#
# JMX management host IP address
```

```
# If not specified (recommended), the first non-local IP address (i.e
   . neither
# 127.0.0.1 nor localhost) on this machine will be used. If no non-
   local IP is
# found, localhost will be used.
#-------------------------------------------------------------#

#jppf.management.host = localhost

#-------------------------------------------------------------#
# JMX management port
# default value is 11198; uncomment to specify a different value
# 2 or more JPPF components on the same machine must use distinct
   values
#-------------------------------------------------------------#

#jppf.management.port = 11198

#-------------------------------------------------------------#
# Internal RMI port used by JMX management
# default value is 12198; uncomment to specify a different value
# 2 or more JPPF components on the same machine must use distinct
   values
#-------------------------------------------------------------#

#jppf.management.rmi.port = 12198

#-------------------------------------------------------------#
# Enable/Disable automatic discovery of JPPF drivers.
# default value is true; uncomment to specify a different value
#-------------------------------------------------------------#

#jppf.discovery.enabled = true

#-------------------------------------------------------------#
# UDP multicast group to which drivers broadcast their connection
   parameters
# and to which clients and nodes listen. Default value is 230.0.0.1
#-------------------------------------------------------------#

#jppf.discovery.group = 230.0.0.1
```

```
#--------------------------------------------------------------#
# UDP multicast port to which drivers broadcast their connection
   parameters
# and to which clients and nodes listen. Default value is 11111
#--------------------------------------------------------------#

#jppf.discovery.port = 11111

#--------------------------------------------------------------#
# How long a driver should wait between 2 broadcasts, in milliseconds
# Default value is 1000
#--------------------------------------------------------------#

#jppf.discovery.broadcast.interval = 1000

#--------------------------------------------------------------#
# Enable/disable auto-discovery for peer-to-peer communication
   between drivers
# Default value is false
#--------------------------------------------------------------#

#jppf.peer.discovery.enabled = true

#--------------------------------------------------------------#
# the name of the load-balancing algorithm to use
# pre-defined possible values are: manual | autotuned | proportional
   | rl
# it can also be the name of a user-defined algorithm (since 2.0)
# default value is "manual"
#--------------------------------------------------------------#
jppf.load.balancing.algorithm = proportional

#--------------------------------------------------------------#
# name of the set of parameter values (aka profile) to use for the
   algorithm
#--------------------------------------------------------------#
jppf.load.balancing.strategy = test

# "manual" profile
strategy.manual.size = 1
```

```
# "autotuned" profile
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

# "proportional" profile
strategy.proportional.size = 5
strategy.proportional.performanceCacheSize = 3000
strategy.proportional.proportionalityFactor = 2

# "rl" profile
strategy.rl.performanceCacheSize = 3000
strategy.rl.performanceVariationThreshold = 0.001

# "test" profile
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 = 3000
strategy.test.proportionalityFactor = 2
strategy.test.increaseRate = 0.03
strategy.test.rateOfChange = 0.9
strategy.test.discountFactor = 0.2
strategy.test.performanceVariationThreshold = 0.001
strategy.test.maxActionRange = 10

#-------------------------------------------------------------#
# Other JVM options added to the java command line when the node is
   started as #
# a subprocess. Multiple options are separated by spaces.
#-------------------------------------------------------------#
```

```
jppf.jvm.options = -Xmx256m

# example with remote debugging options
#jppf.jvm.options = -server -Xmx256m -Xrunjdwp:transport=dt_socket,
   address=localhost:8000,server=y,suspend=n

#---------------------------------------------------------#
# Specify alternate object streams builder for serialization.
# The default value is org.jppf.serialization.
   JPPFObjectStreamBuilderImpl      #
#---------------------------------------------------------#

#jppf.object.stream.builder = org.jppf.serialization.
   XstreamObjectStreamBuilder

#---------------------------------------------------------#
# Specify alternate object stream classes for serialization.
# Defaults to java.io.ObjectInputStream and java.io.
   ObjectOutputStream.        #
#---------------------------------------------------------#

#jppf.object.input.stream.class = java.io.ObjectInputStream
#jppf.object.output.stream.class = java.io.ObjectOutputStream

#---------------------------------------------------------#
# Specify a data transformation class.
# If left unspecified, no transformation is used.
#---------------------------------------------------------#

#jppf.data.transform.class = org.jppf.data.transform.
   DESCipherTransform
```

C.3. Nodo

```
#---------------------------------------------------------#
# JPPF
# Copyright (C) 2005-2010 JPPF Team.
# http://www.jppf.org
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
```

```
# You may obtain a copy of the License at
#      http://www.apache.org/licenses/LICENSE-2.0
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#---------------------------------------------------------------#

#---------------------------------------------------------------#
# Host name, or ip address, of the host the JPPF driver is running on
#---------------------------------------------------------------#

jppf.server.host = localhost

#---------------------------------------------------------------#
# port number for the class server that performs remote class loading
# default value is 11111; uncomment to specify a different value
#---------------------------------------------------------------#

class.server.port = 11111

#---------------------------------------------------------------#
# port number the clients / applications connect to
# default value is 11112; uncomment to specify a different value
#---------------------------------------------------------------#

app.server.port = 11112

#---------------------------------------------------------------#
# port number the nodes connect to
# default value is 11113; uncomment to specify a different value
#---------------------------------------------------------------#

node.server.port = 11113

#---------------------------------------------------------------#
# Enabling JMX features
# default value is true; uncomment to specify a different value
#---------------------------------------------------------------#
```

```
#jppf.management.enabled = true

#--------------------------------------------------------------#
# JMX management host IP address
# If not specified (recommended), the first non-local IP address (i.e
   . neither
# 127.0.0.1 nor localhost) on this machine will be used. If no non-
   local IP is
# found, localhost will be used.
#--------------------------------------------------------------#

#jppf.management.host = localhost

#--------------------------------------------------------------#
# JMX management port
# default value is 11198; uncomment to specify a different value
# 2 or more JPPF components on the same machine must use distinct
   values
#--------------------------------------------------------------#

jppf.management.port = 12001

#--------------------------------------------------------------#
# Internal RMI port used by JMX management
# default value is 12198; uncomment to specify a different value
# 2 or more JPPF components on the same machine must use distinct
   values
#--------------------------------------------------------------#

jppf.management.rmi.port = 13001

#--------------------------------------------------------------#
# path to the JPPF security policy file
# comment out this entry to disable security on the node
#--------------------------------------------------------------#

#jppf.policy.file = config/jppf.policy

#--------------------------------------------------------------#
# Enable/Disable automatic discovery of JPPF drivers.
```

```
# default value is true; uncomment to specify a different value
#-------------------------------------------------------------#

jppf.discovery.enabled = true

#-------------------------------------------------------------#
# UDP multicast group to which drivers broadcast their connection
   parameters
# and to which clients and nodes listen. Default value is 230.0.0.1
#-------------------------------------------------------------#

jppf.discovery.group = 230.0.0.1

#-------------------------------------------------------------#
# UDP multicast port to which drivers broadcast their connection
   parameters
# and to which clients and nodes listen. Default value is 11111
#-------------------------------------------------------------#

jppf.discovery.port = 11111

#-------------------------------------------------------------#
# How long the  node will attempt to automatically discover a driver
   before
# falling back to the parameters specified in this configuration file
# Default value is 5000 miliseconds
#-------------------------------------------------------------#

#jppf.discovery.timeout = 5000

#-------------------------------------------------------------#
# Automatic recovery: number of seconds before the first reconnection
    attempt.
# default value is 1; uncomment to specify a different value
#-------------------------------------------------------------#

#reconnect.initial.delay = 1

#-------------------------------------------------------------#
# Automatic recovery: time after which the system stops trying to
   reconnect,
```

```
# in seconds.
# default value is 60; uncomment to specify a different value
#-------------------------------------------------------------#

reconnect.max.time = 900

#-------------------------------------------------------------#
# Automatic recovery: time between two connection attempts, in
   seconds.
# default value is 1; uncomment to specify a different value
#-------------------------------------------------------------#

#reconnect.interval = 1

#-------------------------------------------------------------#
# Processing Threads: number of threads running tasks in this node.
# default value is 1; uncomment to specify a different value
# blocking tasks might benefit from a number larger then CPUs
#-------------------------------------------------------------#

processing.threads = 1

#-------------------------------------------------------------#
# Specify alternate object streams builder for serialization.
# The default value is org.jppf.serialization.
   JPPFObjectStreamBuilderImpl
#-------------------------------------------------------------#

#jppf.object.stream.builder = org.jppf.serialization.
   XstreamObjectStreamBuilder

#-------------------------------------------------------------#
# Specify alternate object stream classes for serialization.
# Defaults to java.io.ObjectInputStream and java.io.
   ObjectOutputStream.
#-------------------------------------------------------------#

#jppf.object.input.stream.class = java.io.ObjectInputStream
#jppf.object.output.stream.class = java.io.ObjectOutputStream

#-------------------------------------------------------------#
```

```
# Other JVM options added to the java command line when the node is
   started as
# a subprocess. Multiple options are separated by spaces.
#------------------------------------------------------------#

jppf.jvm.options = -Xmx128m

# example with remote debugging options
#jppf.jvm.options = -server -Xmx128m -Xrunjdwp:transport=dt_socket,
   address=localhost:8000,server=y,suspend=n

#------------------------------------------------------------#
# Specify a data transformation class.
# If left unspecified, no transformation is used.
#------------------------------------------------------------#

#jppf.data.transform.class = org.jppf.data.transform.
   DESCipherTransform
```

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and

redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A

Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. page

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Bibliografía

Sueldos y salarios - [En línea] [Citado el: 20 de Septiembre de 2010.] http://www.tufuncion.com/trabajo-programador

JPPF - [En línea] [Citado el: 16 de Septiembre de 2009.] http://www.jppf.org/

JASYPT - [En línea] [Citado el: 13 de Febrero de 2010.] http://www.jasypt.org/index.html

Hazelcast -[En línea] [Citado el: 23 de Enero de 2010.] http://www.hazelcast.com/

NetBeans - [En línea] [Citado el: 7 de Febrero de 2010.] http://netbeans.org/

MySQL - [En línea] [Citado el: 5 de Abril de 2010.] http://www.mysql.com/

Miktext - [En línea] [Citado el: 20 de Septiembre de 2010.] http://miktex.org/

IP-Sniffer - [En línea] [Citado el: 20 de Octubre de 2009.] http://www.ip-sniffer.com/

Ghostscript, Ghostview y GSview - [En línea] [Citado el: 20 de Septiembre de 2010.] http://pages.cs.wisc.edu/~ghost/

Gimp - [En línea] [Citado el: 20 de Septiembre de 2010.] http://www.gimp.org.es/

Pixlr - Tratamiento de imagines online [En línea] [Citado el: 4 de Marzo de 2010 http://www.pixlr.com/

Precio de Microsoft Visio 2010 y Microsoft Project 2010 - [En línea] [Citado el: 24 de Septiembre de 2010.] http://office.microsoft.com/es-es/products/?CTT=97

Apache ant - [En línea] [Citado el: 7 de Octubre de 2009.] http://ant.apache.org/

Apache commons - [En línea] [Citado el: 24 de Abril de 2010.] http://commons.apache.org/

API de Java - [En línea] [Citado el: 13 de Septiembre de 2009.] http://download.oracle.com/javase/1.5.0/docs/api/

Creating a GUI With JFC/Swing: Table of Contents [En línea] [Citado el: 12 de Abril de 2010.] http://download.oracle.com/javase/tutorial/uiswing/TOC.html

JavaTM Cryptography Architecture - [En línea] [Citado el: 12 de Enero de 2010.] http://download-llnw.oracle.com/javase/1.5.0/docs/guide/security/CryptoSpec.html#AppA

Pool de conexiones - [En línea] [Citado el: 7 de Abril de 2010.] http://chuwiki.chuidiang.org/index.php?title=Pool_de_conexiones

Wikipedia - Agrupamiento de conexiones [En línea] [Citado el: 15 de Agosto de 2010.] http://es.wikipedia.org/wiki/Connection_pool

Wikipedia - Clúster [En línea] [Citado el: 8 de Noviembre de 2009.] http://es.wikipedia.org/wiki/Cluster_%28inform%C3%A1tica%29

Wikipedia - Computación distribuida [En línea] [Citado el: 6 de Septiembre de 2010.] http://es.wikipedia.org/wiki/Computaci%C3%B3n_distribuida

Wikipedia - Criptografía [En línea] [Citado el: 9 de Abril de 2010.] http://es.wikipedia.org/wiki/Criptograf%C3%ADa

Wikipedia - Protocolos TCP/IP [En línea] [Citado el: 23 de Enero de 2010.] http://es.wikipedia.org/wiki/Modelo_TCP/IP

Wikipedia - Sistemas de numeración [En línea] [Citado el: 4 de Octubre de 2010.] http://es.wikipedia.org/wiki/Sistema_de_numeraci%C3%B3n

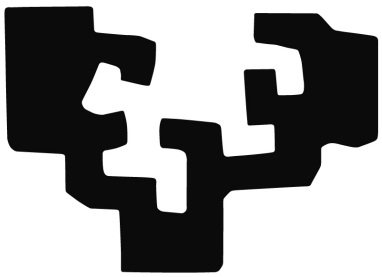
Wikipedia - Swing Java [En línea] [Citado el: 12 de Agosto de 2010.] http://es.wikipedia.org/wiki/Swing_%28biblioteca_gr%C3%A1fica%29

Variaciones con repetición [En línea] [Citado el: 13 de Abril de 2010.] http://thales.cica.es/rd/Recursos/rd99/ed99-0516-02/practica/var_rept.htm

SwingWorker - [En línea] [Citado el: 14 de Mayo de 2010.] https://swingworker.dev.java.net/

Licencias - [En línea] [Citado el: 20 de Septiembre de 2010.] http://www.gnu.org/licenses/licenses.es.html

eman ta zabal zazu



Universidad del País Vasco

Euskal Herriko Unibertsitatea