Universidad Zaragoza

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Proyecto Final de Carrera
Ingeniería Informática
Curso 2011-2012

Diseño e implementación de una capa de red P2P jerárquica, escalable y tolerante a fallos

Víctor Miguel Catalán Sánchez

Septiembre de 2012

Director: Javier Celaya Alastrué

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

A mi tutor, por su respaldo y dedicación durante este tiempo.
A mi madre, por su apoyo incondicional, y su entrega de toda una vida
A mi familia, por apoyarme en todo momento.
A mis amigos y compañeros por todas las ayudas recibidas.
Y a mi novia, por ser como eres y estar siempre a mi lado.

Diseño e implementación de una capa de red P2P jerárquica, escalable y tolerante a fallos.

Resumen

Durante este proyecto se ha diseñado e implementado una capa de red peer-to-peer jerárquica, escalable y tolerante a fallos. Dicha capa es un medio que nos permite la distribución de la información y la localización de recursos a emplear por otras plataformas. Esta capa está enmarcada en el desarrollo de una plataforma escalable para la ejecución distribuida de tareas.

Se ha utilizado una estructura principal en forma de árbol binario balanceado o AVL. Las búsquedas en este tipo de árboles tienen una complejidad que se mantiene siempre en orden logarítmico O(log n), por lo que es perfecta para cumplir el requisito de la escalabilidad. Otro punto fundamental es la tolerancia a fallos, en cuyo caso se resuelve utilizando una DHT como estructura paralela, donde se almacenará la información actualizada de todos los nodos de la red. En caso de fallo de un nodo, cualquier participante puede obtener la información relativa a dicho nodo e iniciar el proceso de reconstrucción que permite devolver a la red a un estado correcto. Entre las distintas opciones de DHT, se ha elegido Apache Cassandra y más en concreto la librería libQtCassandra, que es un cliente que proporciona una API para interaccionar con los nodos de Cassandra.

En definitiva, se ha desarrollado un protocolo de comunicación totalmente distribuido, que permite trabajar de manera conjunta a la estructura en forma de árbol y la DHT. El lenguaje utilizado es C++ por ser el lenguaje que se usa en la plataforma en la que este proyecto está enmarcado.

Para comprobar el buen funcionamiento, se ha utilizado un simulador de eventos discretos. Con el objetivo de comprobar la escalabilidad, se han simulado redes de tamaños desde 10 nodos hasta 500.000 nodos. Posteriormente, se vuelven a simular los mismos tamaños y además se introducen fallos de nodos físicos aleatorios para comprobar la tolerancia a fallos. Una vez concluye la simulación, se comprueba el resultado por medio de pruebas de validación del estado de los participantes de la red. El resultado obtenido tras la realización de todas las pruebas con el simulador es el correcto, con lo que se puede afirmar que la red se adapta perfectamente a cualquier tamaño de red y además detecta los fallos que se producen para posteriormente reconstruir la red. Además se han obtenido medidas del consumo del ancho de banda de entrada y de salida obteniendo valores muy buenos y casi despreciables. También se ha medido el tiempo de inserción de un nodo en la red y se puede concluir que para redes grandes de 100.000 nodos en adelante, el tiempo se estabiliza alrededor de 2,25 segundos, siendo un valor que se considera aceptable.

Asimismo se ha ejecutado la aplicación en un grupo de ordenadores del laboratorio y se ha conseguido interconectarlos. La red se ha creado y funciona correctamente, por lo que la prueba en un escenario real también es un éxito.

Índice

Índice de figuras

1. Introducción

Una red peer-to-peer (a partir de ahora P2P) es una red de computadoras en la que todos los nodos participantes tienen las mismas responsabilidades, es decir, todos los nodos se comportan como iguales entre sí. Un tipo de estructura para una red P2P muy importante es la estructura jerárquica o estructura en árbol. Este tipo es un modelo en el que los diferentes nodos se organizan en múltiples niveles de acuerdo a una estricta relación Padre-Hijo. Un nodo padre puede tener más de un nodo hijo, estando todos ellos localizados en el mismo nivel inmediatamente inferior, pero un nodo hijo únicamente puede tener un nodo padre situado en el nivel inmediatamente superior.

Las estructuras jerárquicas están muy extendidas. Existen multitud de aplicaciones como sistemas de indexación multidimensional como VBI-tree [9], BATON [8] o estructuras jerárquicas de datos como la organización de bases de datos [17].

Otro ejemplo muy común es usar este tipo de estructuras para agregar información. La información fluye desde las hojas del árbol hacia arriba, por lo tanto, la agregación de la información permite a los nodos superiores tener una visión global de las descripciones de todos los nodos situados en sus niveles inferiores. De esta forma, las búsquedas en esta estructura se realizan de forma eficiente, recorriendo un número reducido de nodos gracias a la agregación de la información. Este concepto de agregación de la información se puede estudiar en Astrolable [19] y en [20].

No obstante, el gran número de nodos que pueden formar parte de la red hace que la posibilidad de fallo sea elevada. Sin embargo, cada nodo perteneciente a la estructura jerárquica tiene enlaces únicamente con su padre y sus hijos directos. Combinando estas características se reduce el número de enlaces a restablecer ante un fallo, haciendo esta estructura ideal frente a la escalabilidad y tolerancia a fallos.

Por tanto, para este proyecto fin de carrera se ha decidido desarrollar una capa de red P2P jerárquica, escalable y tolerante a fallos. Esta estructura se integra dentro de una plataforma escalable para la ejecución distribuida de tareas [6], que la utiliza como medio para la distribución de información y localización de recursos.

1.1. Objetivos

El objetivo principal de este proyecto fin de carrera es el diseño e implementación de una capa de red peer-to-peer jerárquica, escalable y tolerante a fallos.

Dicha capa de red se sitúa dentro del desarrollo de una plataforma encargada de la planificación distribuida de trabajos o tareas puramente de cálculo y no interactivas. Esta plataforma, para llevar a cabo su objetivo, requiere de un medio para la distribución de la información y la localización de recursos a emplear, por lo que una capa de red jerárquica parece la mejor solución a implementar.

A esta capa de red se le exige escalabilidad, entendida como su capacidad para adaptarse a nuevos tamaños con un mínimo impacto en su desempeño, así como la capacidad parar tolerar y recuperarse del fallo de una cierta cantidad de nodos participantes en la red sin afectar a los demás miembros.

Para la realización satisfactoria de este trabajo se estudiará el estado del arte sobre el diseño de las capas de red P2P en general, y jerárquicas en particular, comparando sus propiedades de escalabilidad y tolerancia a fallos, eligiendo la que más se adecue a nuestro escenario. Se diseñará una capa de red que cumpla con los requisitos expuestos con anterioridad y posteriormente se implementará sobre la plataforma. Finalmente se comprobará mediante simulación el correcto funcionamiento de la estructura y además se estudiará en qué medida se ha cumplido con los objetivos de escalabilidad y tolerancia a fallos.

1.2. Organización de este documento

A continuación este trabajo está dividido en los siguientes capítulos:

- Capítulo 2 - Proporciona una visión de las opciones y las decisiones tomadas.
- Capítulo 3 - Detalla la arquitectura de la red y los detalles concretos de diseño.
- Capítulo 4 - Describe el entorno de programación y medidas del software.
- Capítulo 5 - Describe el entorno de simulación y las pruebas realizadas.
- Capítulo 6 - Analiza los resultados y una posible continuación futura.

Al final se incluye también los anexos que completan este trabajo en el que podemos encontrar: *Despliegue Apache Cassandra* (Anexo A) *Código fuente* (Anexo B) y *Licencia GPL* (Anexo C).

2. Análisis del problema

En este proyecto se plantea el problema que hace referencia a un conjunto de computadores conectados entre sí formando una red P2P. Se trata de crear una capa de red P2P que proporcione conectividad y permita que los datos, de una aplicación que se base en esta capa, lleguen desde un computador origen a un computador destino aunque no tengan conexión directa.

La capa de red P2P deber tener las siguientes características: jerárquica, escalable y tolerante a fallos. Las dos primeras características se describen en el apartado 2.1, mientras que en el aparatado 2.2 se discuten posibles soluciones a la tolerancia a fallos. Por último, la interfaz externa se describe en el apartado 2.3.

2.1. Arquitectura de la red

Uno de los requisitos de este proyecto es utilizar un estructura en forma de árbol y más concretamente, un árbol binario balanceado o AVL. Por tratarse de una estructura en árbol se hablará indistintamente de nodos o participantes en la red. Todos los nodos de los árboles AVL están siempre equilibrados, es decir, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a este equilibrio la complejidad de una búsqueda se mantiene siempre en orden O(log n).

La escalabilidad, que se entiende como la capacidad que tienen las redes P2P para albergar un número de nodos muy elevado sin afectar el tiempo de respuesta de la red y conservando las prestaciones de la misma, es una característica que se busca en esta estructura. Los árboles AVL cumplen los requisitos de escalabilidad para un número muy grande de participantes debido a la complejidad logarítmica de las búsquedas.

Hay que destacar, que el verdadero desafío de este proyecto es diseñar un protocolo de comunicación que gestione la estructura en forma de árbol, totalmente distribuida y sin memoria compartida. Manteniendo el punto de mira en la escalabilidad, el protocolo debe involucrar el menor número de nodos posible a la vez, así como reducir en la medida de lo posible el tráfico generado por los participantes.

2.2. Tolerancia a fallos

La tolerancia a fallos es un punto crucial en una red P2P para que ésta no pierda su funcionalidad, debido a fallos asociados a desconexiones o caídas de la red. Por este motivo, se ha pensado en una estructura paralela sobre la que se construya el árbol AVL y a la que se accede en caso de fallo. La estructura paralela utilizada es una tabla hash distribuida (a partir de ahora DHT por sus siglas en inglés)[10]. Es una estructura descentralizada y escalable, que nos permite almacenar y recuperar de manera eficiente la información mediante el par Clave-Valor, y que además, cuenta con sus propios mecanismos de tolerancia a fallos como la replicación.

En la DHT se guarda el estado de cada nodo en cada instante. Cuando un nodo modifica su información se actualiza también en la DHT, manteniendo la consistencia entre la estructura AVL y la DHT. En el momento que un nodo falla, el resto de nodos de la red pueden acceder de forma eficiente a la DHT y así pueden obtener la información de dicho nodo que ha fallado e iniciar la reconstrucción de la estructura.

Algunos ejemplos de DHT son CAN [15], Chord [18], Tapestry [21], Kademlia [12], o Pastry [16]. Entre los ejemplos anteriores encontramos el proyecto maidsafe [3]. Es una DHT para redes P2P descentralizadas que implementa el protocolo Kademlia. Es un proyecto público y desarrollado en C++, que se adapta perfectamente a nuestros requisitos. Sin embargo, se desestima por haberse paralizado sin liberar una versión estable.

Otra estructura a destacar, que no es exactamente una DHT pero actúa y tiene las mismas características, es Apache Cassandra [1]. Es una base de datos no relacional, distribuida y basada en un modelo de almacenamiento del par Clave-Valor, que permite almacenar y gestionar grandes volúmenes de datos de forma distribuida. Es decir, Apache Cassandra actúa como las DHTs y, además, se adapta a la perfección a los requisitos de la capa de red. Este proyecto está desarrollado por Apache Software Foundation [4], lo cual nos otorga una garantía en la madurez y seriedad del proyecto. Esta aplicación es utilizada por numerosas empresas tecnológicas de alto renombre como Facebook y Twitter.

Debido a todo lo anterior, se selecciona como estructura paralela para resolver la tolerancia a fallos al proyecto Apache Cassandra. Más en concreto, se usará la librería libQtCassandra [5]. Es un cliente de alto nivel que proporciona una API para interaccionar con los servidores de Cassandra en lenguaje C++.

2.3. Interfaz externa

El objetivo de este proyecto es la integración de una capa árbol, con una plataforma encargada de la planificación distribuida de trabajos. Uno de los requisitos es que la capa Árbol implemente una interfaz externa que pueda ser usada por el resto de capas para realizar sus tareas.

El modelo de capas que representa este proyecto aparece en la figura 2.1.

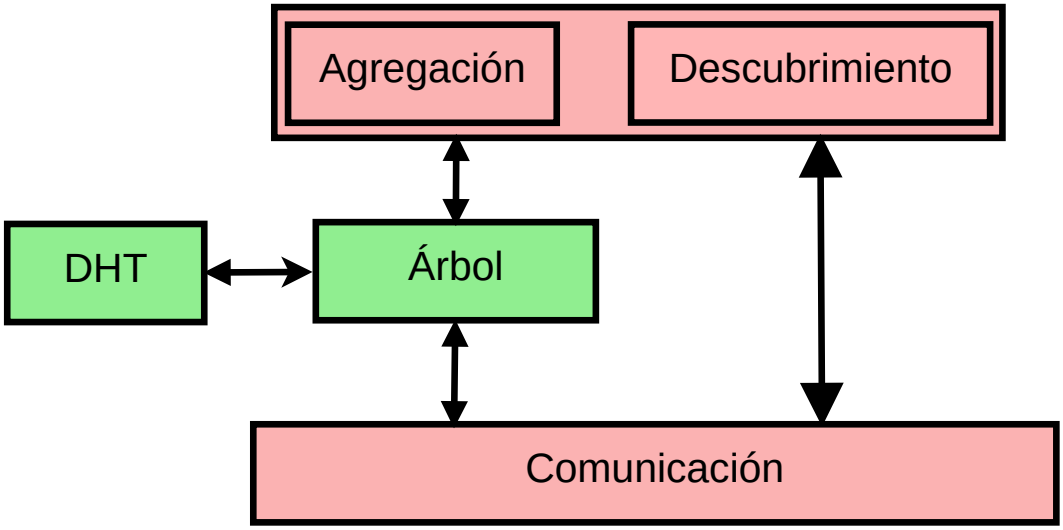

Figura 2.1: Visión global de todo el proyecto por capas

Se puede observar la capa de comunicación que engloba todo lo relacionado con las conexiones de red, como enviar y recibir mensajes y todo el tratamiento de fallos en la red. Dicha capa nos permite interconectar a todos los participantes y que puedan intercambiar mensajes entre sí.

Las capas de árbol y DHT son las que se han desarrollado en este proyecto. La capa árbol gestiona cómo se estructuran todos los participantes en la red, determinando cual es la posición de cada nodo en el árbol. También debe informar al resto de las capas, los cambios en la estructura que se producen. Por otro lado, la capa DHT es la encargada de almacenar la información necesaria de los nodos, para dotar a la estructura de tolerancia a fallos.

Otras capas de más alto nivel, son las de agregación y descubrimiento. La capa de agregación permite agregar la información de los nodos suministrada por la capa árbol. A su vez, también permite a la capa de descubrimiento encontrar dónde están los recursos que se necesitan. Estas dos capas requieren de la interfaz suministrada por la capa árbol para poder ejecutar sus funciones.

3. Diseño de la solución

A partir de las decisiones que se han tomado en el capítulo de análisis, se va a detallar la estructura jerárquica, el mantenimiento de dicha estructura y la conectividad entre los nodos participantes en la red.

3.1. Arquitectura de la red

La estructura y la gestión del árbol que conforma la arquitectura de la red se deriva del trabajo sobre árboles binarios balanceados, también llamados árboles AVL[2] aunque con ciertas modificaciones. En primer lugar, el par de la dirección IP-Puerto será el valor identificativo que represente a cada nodo físico de forma unívoca en la red. Este par identificativo se utiliza también como clave para acceder a los datos de la DHT.

En la estructura jerárquica cada nodo físico interpreta dos roles, un nodo con rol de estructura (Structure Node) y un nodo con rol de recurso (Resource Node). Cada rol que interpreta un nodo físico está localizado en una posición dentro del árbol, destacando que dichas posiciones no guardan ninguna relación entre sí.

Los nodos con rol de estructura tienen un enlace con su padre y un enlace con cada uno de sus hijos, izquierdo y derecho. Además guarda la descripción de zona de cada hijo, es decir, la información relativa a cada subárbol, que permitirá encaminar los mensajes. La descripción de zona contiene la dirección mínima y máxima, la altura y qué rama es la más alta para cada subárbol. Por otra parte, los nodos de recurso tienen un único enlace con su padre. Existe un cambio importante ya que los hijos de los nodos de estructura no se sitúan a la izquierda o a la derecha de un valor. Cada nodo de estructura guarda el intervalo de direcciones de sus subárboles que representan las tablas de encaminamiento para las operaciones de la red. En la figura 3.1 se puede observar más claramente los detalles.

Los nodos con rol de estructura serán los nodos internos en el árbol y los que nos permitirán ir recorriendo el árbol hasta encontrar el recurso que se esté buscando. Siempre tendrán hijos ya sean dos nodos de estructura, dos nodos de recurso o un nodo de estructura y otro de recurso. Los nodos con rol de recurso siempre se encontrarán en las

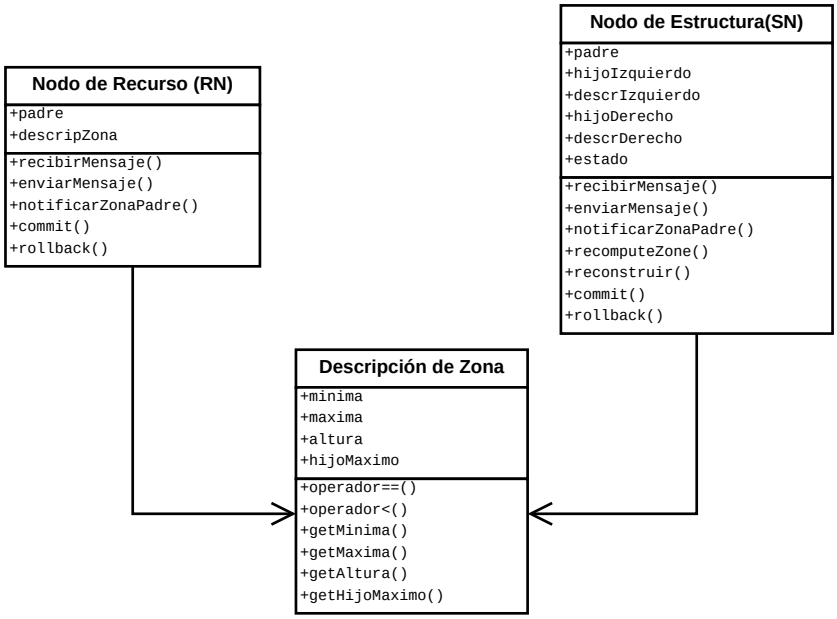

Figura 3.1: Diagrama de clases de los nodos de estructura, de recurso y de la descripción de zona

hojas inferiores del árbol y su padre siempre será un nodo interno de estructura. Esta división en roles permite separar el mantenimiento de la estructura, que se produce en los nodos de estructura, del uso del recurso perteneciente a cada nodo.

En la figura 3.2 se pueden ver las diferencias entre un árbol AVL y el diseñado, teniendo en cuenta que los rectángulos representan los nodos de estructura y los círculos los nodos de recurso, así como la DHT que almacena la información. En el árbol diseñado siempre se

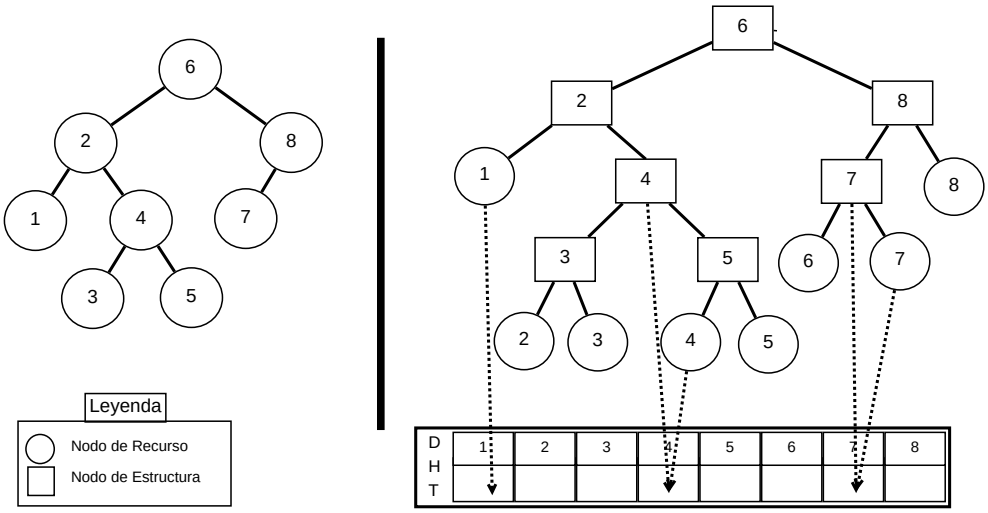

Figura 3.2: Esquema diferencial entre un árbol AVL y su correspondiente nuevo árbol diseñado.

cumplirá que el número de nodos totales (suma de los que interpretan el rol de estructura y de recurso) es el doble de nodos físicos menos uno. Esto es debido a que por cada nodo

físico siempre se introduce un nodo interpretando el rol de estructura y otro nodo con el rol de recurso salvo el primer nodo que solo introduce un nodo con el rol de recurso, ya que la red es el propio nodo de recurso.

3.2. Operaciones de la red

Existen tres operaciones relacionadas con la estructura de la red:

- La inserción, que permite añadir nuevos nodos a la red.
- Las rotaciones, que permiten cambiar la estructura del árbol sin interferir en el orden de los elementos modificados haciendo que los nodos estén balanceados.
- La gestión de fallos, que nos permite reconstruir la estructura de la red ante fallos.

Existen otras operaciones más simples que ayudan a mantener la estructura como son las actualizaciones de zona y la comprobación de la diferencia de altura.

3.2.1. Concurrencia

En entornos distribuidos, poder realizar múltiples operaciones al mismo tiempo es vital. Sin embargo, cuando varias operaciones utilizan los mismos recursos hay que organizar cómo se llevan a cabo todas las operaciones sin interferir en las demás. Esta organización permite mantener la consistencia de la red y es lo que se denomina concurrencia de las operaciones. La concurrencia en árboles AVL se ha estudiado ampliamente en [7] [14].

Para resolver la concurrencia de la capa de red se toma la decisión de usar transacciones[11]. Las transacciones nos permiten llevar a cabo una tarea independientemente del resto. Esto nos asegura que la tarea se realice de una sola vez y sin que se altere en medio de una modificación. Cada tarea genera un identificador de transacción único que se envía a los recursos involucrados. Existe un identificador de transacción nulo que representa que un recurso está libre. Por este motivo, antes de empezar una tarea es necesario comprobar si el recurso este libre y al finalizar una tarea hay que liberar el recurso asignando el identificador nulo.

El uso de las transacciones se implementa con el protocolo de commit en dos fases (Two-Phase-Commit)[13]. Es un algoritmo distribuido que permite a todos los nodos participantes ponerse de acuerdo para tomar una decisión. En este algoritmo hay un coordinador que es el que inicia la transacción. Las dos fases del algoritmo son la fase de petición de commit, en la cual el coordinador intenta preparar a todos los demás participantes, y la fase de commit, en la cual el coordinador completa las transacciones a todos los demás. El resultado del protocolo es que todos los nodos realizan el cambio

(Commit) o abortan (Rollback) la transacción. Se ha elegido este protocolo por su sencillez y reducido número de mensajes.

Aparte de lo comentado anteriormente a la hora de tomar las decisiones si un nodo está involucrado en dos transacciones se establece que las inserciones tienen preferencia frente a las rotaciones. Si un nodo está rotando y recibe un mensaje de inserción toma la decisión de abortar la rotación e iniciar la inserción. Esta decisión se debe a que la inserción se realiza en las hojas del árbol, por lo que cuando ésta finalice y se actualicen los nodos afectados y sucesivos hacia arriba, en caso de que sea necesario se iniciará nuevamente la rotación. Las inserciones entre sí no tienen preferencia y se realiza la que llega en primer lugar, retrasando una que llegue más tarde. En cuanto a las rotaciones tiene preferencia una rotación que se inicie más abajo en el árbol. Al finalizar la rotación y se realicen las actualizaciones de los implicados se volverá a iniciar la rotación en caso de ser necesario. En el caso de una reconstrucción, ésta siempre tendrá preferencia sobre el resto, ya que al haber un nodo caído involucrado no se puede llevar a buen término la transacción. En este caso para el correcto funcionamiento y mantenimiento de la consistencia de la estructura, se abortan las transacciones involucradas y se inicia la reconstrucción.

3.2.2. Inserción

La inserción es la operación que permite añadir nuevos nodos a la red. La secuencia de acciones a realizar está descrita en el diagrama de secuencia de la figura 3.3. En el

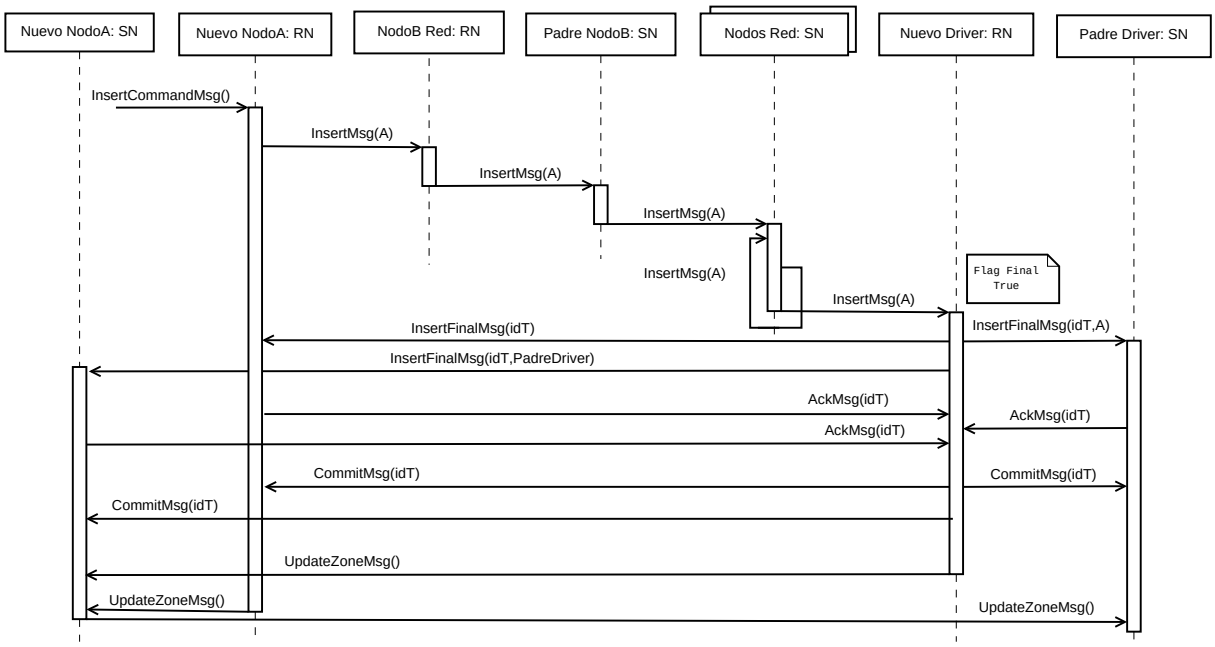

Figura 3.3: Diagrama de secuencia de la inserción. Caso general

caso general, el nuevo nodo (nodo A) que se quiere insertar en la red debe contactar con

algún nodo (nodo B) conocido que ya esté formando parte de la red. El nodo A envía un mensaje de inserción (InsertMsg) al nodo B. Este mensaje lo recibirá un nodo de recurso que lo retransmite hacia arriba, a su padre. El nodo de estructura compara la dirección del nodo A con su descripción de zona. Si la dirección está comprendida en su intervalo de direcciones retransmite el mensaje por la rama que corresponda, izquierda o derecha, o de lo contrario el mensaje es retransmitido al padre. El mensaje va subiendo por los nodos de estructura repitiendo el proceso hasta que la dirección esté comprendida en el intervalo de direcciones o llegue al nodo raíz. Llegados a este punto, el mensaje desciende a través de las ramas adecuadas hasta llegar a la hoja que será un nodo de recurso, que se convierte en el coordinador de la transacción.

El coordinador envía mensajes (InsertFinalMsg) a los nodos implicados para informarles de cuál será su siguiente estado. Los nodos implicados son el coordinador, el padre del coordinador, y el nuevo nodo (nodo A) de estructura y de recurso. Una vez todos se ponen de acuerdo y se realiza la inserción, el coordinador y el nuevo nodo de recurso son hermanos y tienen como padre al nuevo nodo de estructura. A su vez, el nuevo nodo de estructura tiene como padre el que era padre del coordinador, como se puede ver en la figura 3.4 aclaratoria.

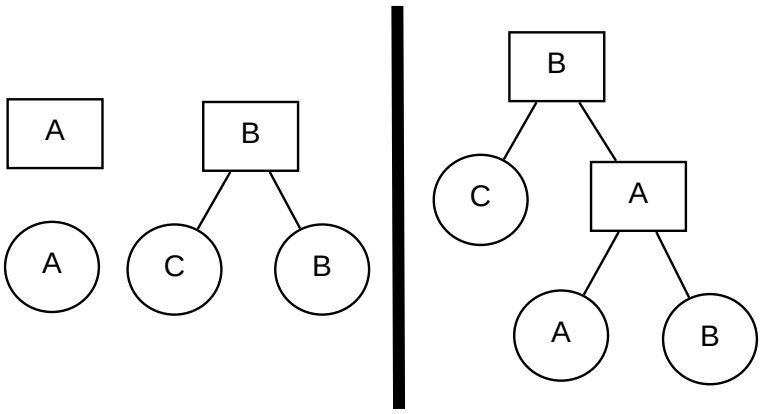

Figura 3.4: Representación del antes y después de la inserción

Tras realizar la inserción, cada nodo involucrado en ella debe actualizar su descripción de zona, y para ello se utilizan los mensajes UpdateZoneMsg. Estos mensajes recorren el árbol desde las hojas hacia arriba según se explica en la sección 3.2.4.

3.2.3. Rotaciones

La rotación es la operación que permite cambiar la estructura del árbol sin interferir en el orden de los elementos modificados. Esta operación se inicia siempre en un nodo de estructura con el objetivo de mantener balanceado dicho nodo. El objetivo es equilibrar la altura de las ramas izquierda y derecha, permitiendo una diferencia máxima de una unidad. El resultado de una rotación es la reestructuración del árbol. Se equilibran las

alturas moviendo los subárboles más pequeños hacia abajo y los subárboles más altos hacia arriba.

Existen cuatro posibles rotaciones que se pueden aplicar. En este apartado se explica cómo se llevan a cabo cada una de las rotaciones, y posteriormente se analiza cuando hay que realizar cada una. Debido a algunas inconsistencias en la definición de las direcciones de las rotaciones, se ha tomado la decisión de definirlas indicando que dirección, derecha o izquierda, de los subárboles es la más alta. Debido a la simetría que existe se pueden dividir en dos grupos.

a) **Rotación Derecha-Derecha (RR) y Rotación Izquierda-Izquierda (LL)**

Las rotaciones RR y LL son simétricas. La rotación RR se usa cuando el subárbol derecho sea dos unidades más alto que el izquierdo, y a su vez el subárbol derecho del hijo derecho sea una unidad más alto que el izquierdo. Llegados a este punto se procede del siguiente modo:

- Pasamos el subárbol izquierdo del hijo derecho como subárbol derecho del nodo raíz de la rotación. Esto mantiene el árbol ordenado, ya que todos los valores del subárbol movido siguen siendo mayores que los que teníamos en el subárbol derecho.
- El nodo raíz de la rotación pasa a ser el subárbol izquierdo del hijo derecho.

Con estas dos simples modificaciones el nuevo árbol queda equilibrado en cuanto a la altura se refiere. De manera simétrica se realiza la rotación LL. Las situaciones iniciales y finales de las dos rotaciones se pueden ver de manera gráfica en la siguiente figura 3.5.

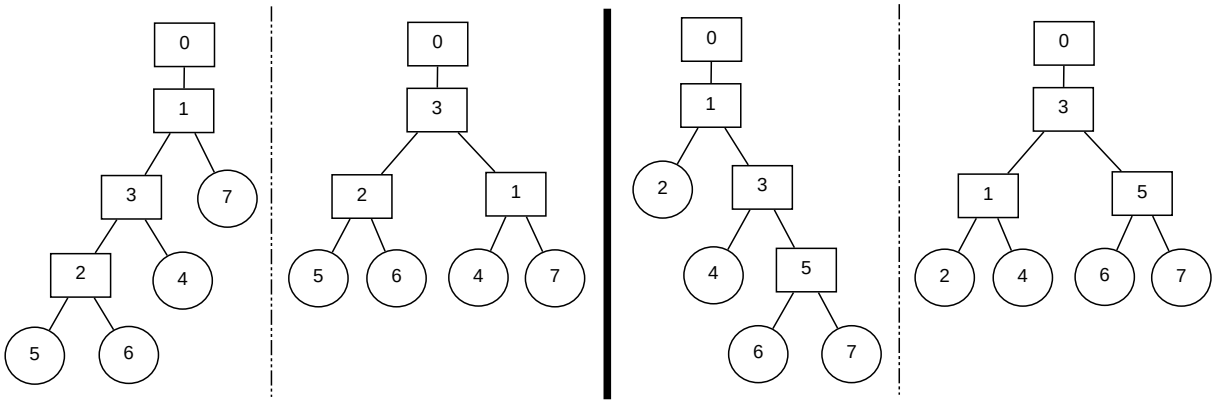

Figura 3.5: Representación del antes y después de la rotación LL y RR

La secuencia de acciones a realizar hasta completar la rotación descrita es la que se puede ver en la figura 3.6. El nodo 1, raíz de la rotación, envía el mensaje RotateMsg con la nueva información a su padre nodo 0 y a su hijo nodo 3. Posteriormente, cuando estos reciben el mensaje, devuelven el mensaje de AckMsg al coordinador como

que han recibido la información nueva y en el caso del nodo 3 nos devuelve además el enlace del hijo que va a formar parte de la rotación, el nodo 4. El coordinador manda un mensaje RotateMsg al nodo 4 con su información y a continuación este nodo 4 contesta con el AckMsg. En el momento en el que el coordinador recibe este mensaje envía los CommitMsg a todos los participantes. Una vez recibido el mensaje se modifica el nodo y pasa a estar en un nuevo estado. Todos los nodos involucrados actualizan su zona desde abajo hacia arriba.

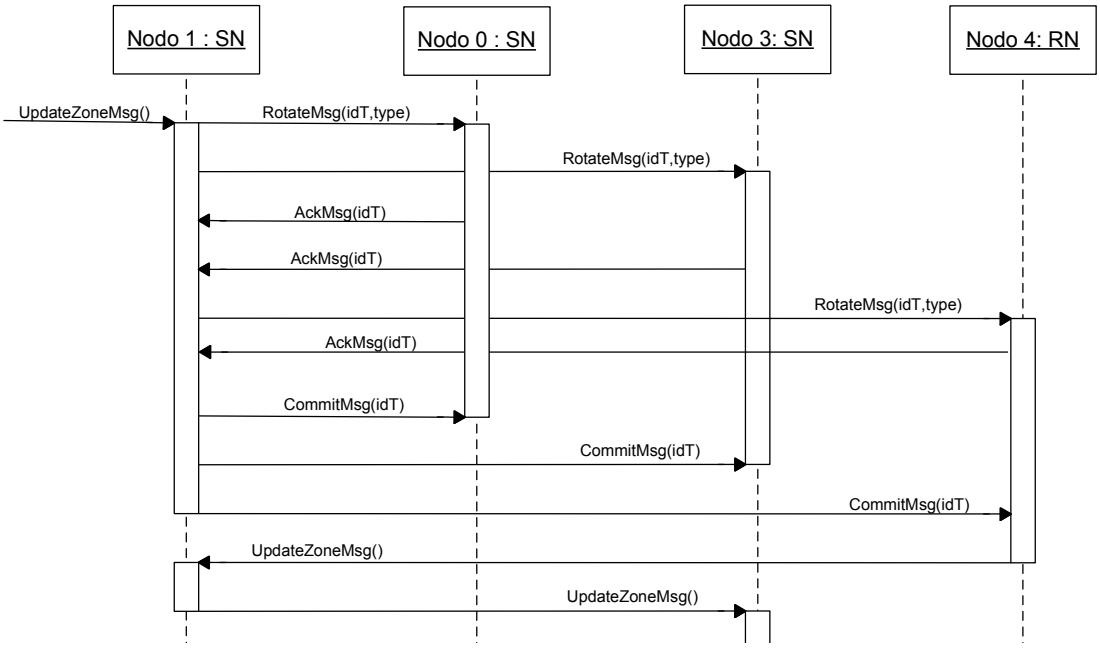

Figura 3.6: Diagrama de secuencia de la rotación LL y RR

b) Rotación Derecha-Izquierda (RL) y Rotación Izquierda-Derecha (LR)

Las rotaciones RL y LR son simétricas. La rotación RL se usa cuando el subárbol derecho de un nodo es dos unidades más alto que el izquierdo y a su vez el subárbol izquierdo del hijo derecho es una unidad más alto que el derecho. Para llevar a cabo estas rotaciones se subdividen en dos pasos. En primer lugar se modifica el árbol realizando una rotación simple del subárbol izquierdo del hijo. Con esta modificación se deja preparado para efectuar posteriormente una rotación de las explicadas anteriormente, en este caso RR. Seguimos los siguientes pasos:

- Pasamos el subárbol derecho del hijo izquierdo del subárbol derecho de la rotación, como subárbol izquierdo del subárbol derecho.
- El hijo izquierdo del subárbol derecho sube un nivel y pasa a ser el hijo del nodo raíz de la rotación y padre del que antes era el subárbol derecho.

Con estas dos simples modificaciones se ha conseguido transformar el árbol a la situación anterior en la que se puede realizar la rotación RR. Las situaciones iniciales

y finales de las dos rotaciones se pueden ver de manera gráfica en la siguiente figura 3.7.

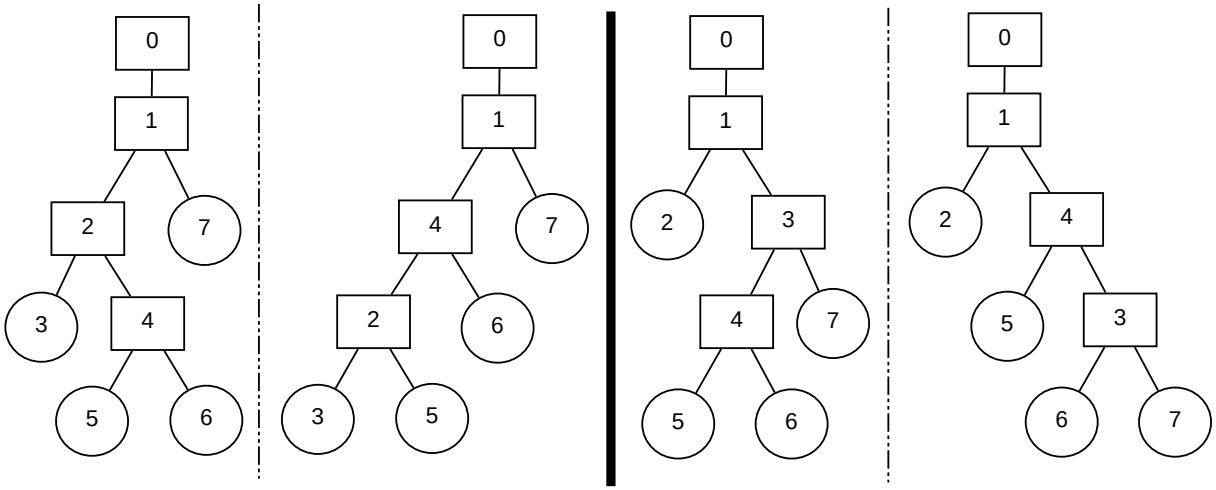

Figura 3.7: Representación del antes y después de la rotación LR y RL

3.2.4. Otras Operaciones

a) **Actualizaciones de zona**

Las actualizaciones de zona son muy importantes, ya que éstas permiten dar a conocer los cambios que se han realizado en cualquier punto del árbol a sus nodos superiores. Se realiza a través del mensaje UpdateZoneMsg, en el que se manda la descripción de zona del propio nodo a su padre. Tras realizar cualquier cambio, ya sea tras inserción, rotación o reconstrucción, todos los nodos que hayan cambiado están obligados a emitir un mensaje de actualización de zona.

Un nodo que recibe una actualización de cualquiera de sus hijos comprueba, y si es necesario, actualiza sus valores de la descripción de zona. Las actualizaciones se seguirán produciendo en orden ascendente hasta llegar al nodo raíz o llegar a un nodo en el que su descripción de zona no se haya modificado.

Los nodos de recurso no tienen ninguna restricción, y una vez hayan cambiado de padre envían el mensaje de actualización instantáneamente. Sin embargo los nodos de estructura tienen restricciones, ya que tras realizar algún cambio es posible que no tengan toda la información. Un nodo de estructura tiene que tener la descripción de zona de sus dos hijos, y hasta que esto no ocurra no puede mandar su mensaje de actualización hacia arriba.

Diferencia de Altura	Subárbol más Alto	Tipo de Rotación
-2 o menor	0 ó Positivo	Right Right
-2 o menor	Negativo	Right Left
2 o mayor	Positivo	Left Right
2 o mayor	0 ó Negativo	Left Left

Tabla 3.1: Valores para tomar decisión de rotación

b) **Comprobación diferencia de altura**

Esta operación es fundamental en el correcto funcionamiento de la estructura, ya que es la que permite comprobar que los nodos estén equilibrados. Esta operación es la que determina si es necesario realizar una rotación, y en ese caso qué tipo de rotación. La comprobación se realiza en los nodos de estructura únicamente, y se lleva a cabo en dos ocasiones: tras cualquier cambio producido por una inserción, rotación o reconstrucción y tras recibir un mensaje de actualización. La diferencia de altura es computada en cada nodo y es definida como la diferencia entre las alturas del subárbol izquierdo menos el derecho. Dada la definición, si la diferencia de altura tiene los valores (-1,0,1), el nodo está equilibrado y por lo tanto no hay que realizar ninguna rotación. Sin embargo si la diferencia de altura es menor o igual a -2 o mayor o igual a 2 hay que realizar una rotación.

Llegados a este punto, hemos decidido si hay que realizar una rotación o no. En caso positivo, hay que determinar qué tipo de rotación y para ello se usará el valor de qué subárbol es el más alto. Este valor si es negativo indica que el subárbol izquierdo es más alto; si es positivo indica que el subárbol derecho es más alto; y si es 0 indica que tienen la misma altura.

Definidos los valores, las decisiones sobre qué rotación se va a realizar aparece en la tabla 3.1.

3.3. Tolerancia a fallos

La DHT actúa de almacén donde se replica toda la información correspondiente a cada nodo. Cada vez que un nodo recibe un mensaje, y éste modifica cualquier información del nodo, la información se actualiza simultáneamente en la DHT. En el momento en que se desconecta o falla un nodo, se rompe la conectividad en una parte de la red. En primer lugar se explican los mecanismos de desbloqueo de nodos involucrados en una operación con un nodo que ha fallado, y posteriormente, cómo se detecta y cómo se reconstruye la red.

3.3.1. Tiempos de espera de las operaciones

Los tiempos de espera nos van a permitir desbloquear los nodos que se queden bloqueados debido al fallo de otro nodo involucrado en la misma transacción. Al realizar cualquier operación, si se supera el tiempo de espera, se considera que ha habido algún problema por lo que se decide abortar la transacción. Los casos son los siguientes:

- Un nodo que intenta acceder a la red manda un mensaje de entrada (InsertCommandMsg). Se activa una alarma al mandar el mensaje. Si se cumple el tiempo de espera de la alarma, dicho mensaje se reenvía para permitir a todos los nodos acceder a la red. Esta alarma tiene una vida activa desde el momento de envío del primer mensaje hasta que el coordinador inicia la inserción. Si se produce algún fallo en la inserción el timer se reactiva.
- Todos los nodos involucrados en una inserción o una rotación en el momento que reciben el mensaje de inicio de transacción activan una alarma para la transacción. Si se cumple el tiempo de espera, la transacción se aborta y se desbloquean los nodos, mientras que en caso de completarse la transacción se desactiva la alarma.
- Para asegurar el correcto balanceo de todos los nodos de la red hay dos situaciones especiales. La primera se produce cuando un nodo de estructura aborta una transacción, y la segunda en el nodo hijo del coordinador de las rotaciones RL y LR. En estas dos situaciones se necesita crear una alarma para asegurarnos que se comprueba el balanceo posteriormente. Si se cumple el tiempo de espera se comprueba si es necesario balancear. La alarma tiene vigencia hasta que se compruebe el balanceo, ya sea por dicha alarma o por el desarrollo normal de la estructura.

3.3.2. Detección de fallos

Todos los nodos cada cierto tiempo, que es fijo, envían un mensaje (AliveMsg) a sus vecinos. Un nodo de estructura envía el mensaje a su padre y a sus dos hijos, mientras que un nodo de recurso sólo envía el mensaje a su padre. Este mensaje (AliveMsg) permite conocer a los nodos que sus vecinos son quienes deben ser y están en la red y no se han desconectado. Cuando un nodo falla deja de enviar los mensajes de AliveMsg.

Todo nodo de la red tiene que recibir este mensaje dentro de un tiempo razonable, que se ha definido como tres veces el intervalo con que se manda el mensaje AliveMsg. Cada vez que un nodo recibe el mensaje de su vecino reinicia el tiempo máximo de espera. Si un nodo no recibe este mensaje de su vecino dentro del tiempo definido, se considera que ese nodo ha fallado.

3.3.3. Elección del coordinador

Un nodo que ha detectado un fallo de algún vecino inicia el proceso de elección del coordinador de la reconstrucción. Este nodo obtiene la información almacenada en la DHT del nodo físico que ha fallado, rol de estructura y rol de recurso. En primer lugar se comprueba si en el momento del fallo estaban involucrados en alguna transacción. En caso de estar en medio de alguna transacción, decide preguntando a los involucrados si la transacción finalizó o por lo contrario abortó y realiza la actualización de los datos en la DHT.

En este momento con los datos actualizados, el nodo que detectó el fallo acude nuevamente a la DHT y obtiene el identificador único del padre y de los dos hijos del nodo de estructura y el identificador del padre del nodo de recurso que ha fallado. Una vez tenemos todos los identificadores se calcula el máximo valor de ellos. El máximo valor es él que se convierte en el coordinador de la reconstrucción.

3.3.4. Reconstrucción de la red

Una vez elegido el coordinador, se inicia el proceso de reconstrucción. En primer lugar, el coordinador comprueba si el nodo que ha fallado estaba interpretando los dos roles o sólo el rol de recurso en la red. A continuación se procede de la siguiente manera:

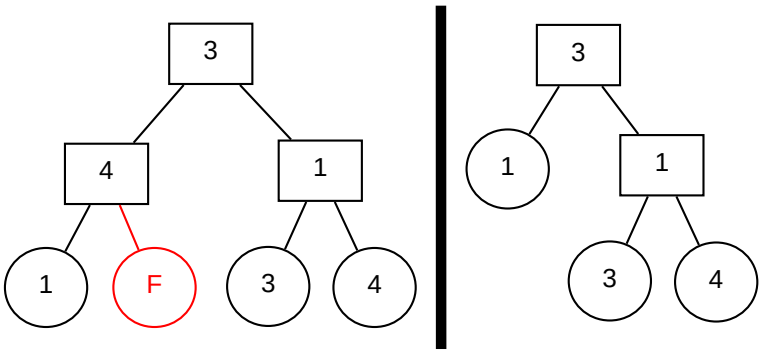

Figura 3.8: Antes y después de reconstrucción de un nodo de recurso

a) **Sólo rol de recurso**

En este punto hay que recordar que en la red siempre hay un nodo menos con el rol de estructura que con el rol de recurso. De este modo, para mantener esta propiedad, otro nodo con el rol de estructura tiene que dejar la red.

Se decide que el padre del nodo que ha fallado, que interpreta el rol de estructura, dejará la red. El hermano del nodo que ha fallado pasará a ocupar el lugar del padre que ha salido de la red, convirtiendose en hijo del nodo que antes era su abuelo.

Este ejemplo se puede observar en la figura 3.8, donde se ve más claramente el antes y el después del fallo del nodo etiquetado "F".

Este fallo es el menos común, ya que solo hay un nodo que este interpretando solamente el rol de recurso.

b) **Ambos roles**

Un nodo ha fallado en dos puntos de la estructura, ya que está interpretando los dos roles. Por lo tanto, el coordinador tiene que resolver consultando en la DHT si el nodo de recurso y el nodo de estructura que han fallado son padre e hijo o no tienen relación.

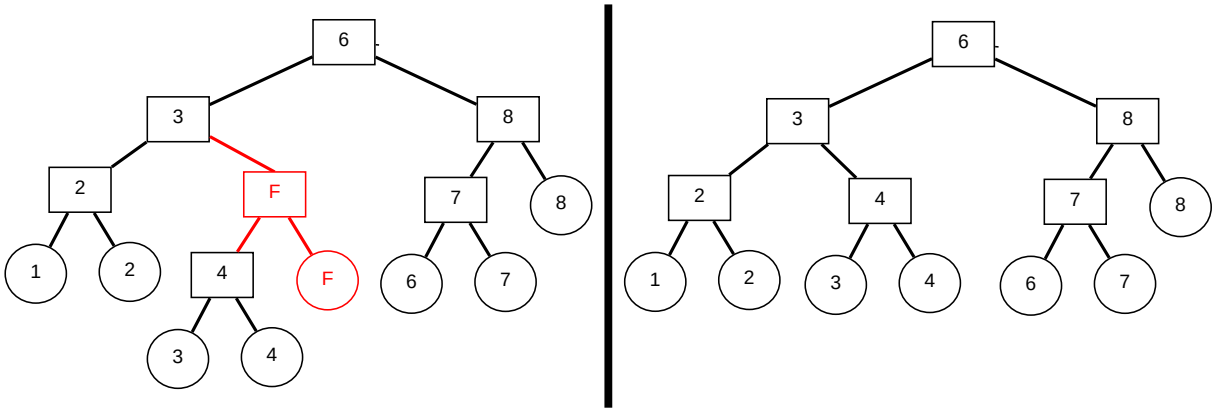

Figura 3.9: Reconstrucción siendo padre e hijo

b.1) **Padre e hijo**

En la reconstrucción el hermano del nodo de recurso que ha fallado, ya sea un nodo de recurso o estructura, pasará a ocupar el lugar del padre que ha fallado de la red, siendo hijo del que era su abuelo. Este ejemplo se puede observar en la figura 3.9, donde se ve más claramente el antes y el después del fallo del nodo etiquetado "F".

b.2) **No tienen relación**

En la reconstrucción el padre del nodo de recurso que ha fallado, que es un nodo de estructura, pasa a sustituir al nodo de estructura que ha fallado. Además por otro lado, el hermano del nodo de recurso que ha fallado, pasa a sustituir a su padre, siendo hijo del que era su abuelo. Este ejemplo se puede observar en la figura 3.10, donde se ve más claramente el antes y el después del fallo del nodo etiquetado "F".

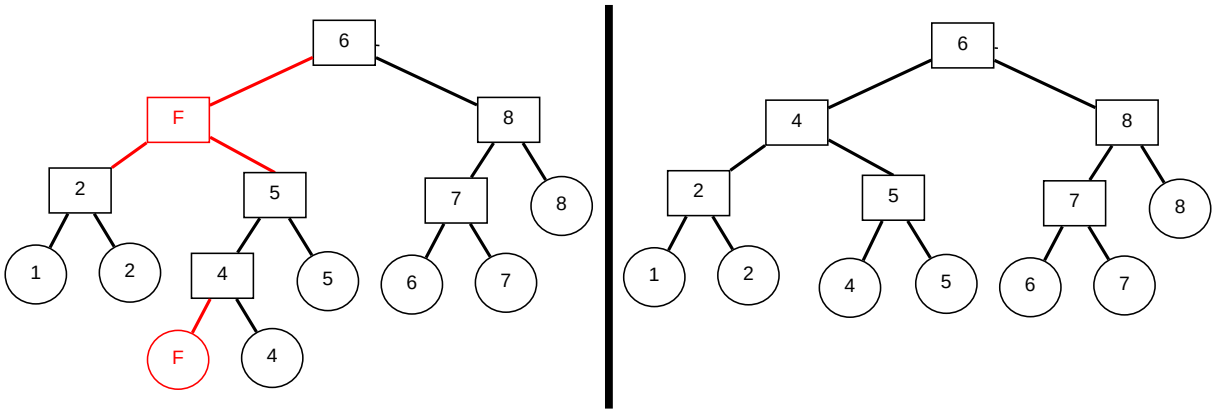

Figura 3.10: Reconstrucción sin ninguna relación

En las tres situaciones, el coordinador envía el mensaje (InitRebuildMsg) a todos los nodos implicados con la nueva información. Tras consolidar los cambios, cada nodo actualiza su descripción de zona y el coordinador borra los datos correspondientes al nodo fallado de la DHT.

4. Implementación de la solución

4.1. Entorno de programación

El lenguaje elegido para realizar la implementación de la capa de red es C++. La plataforma en la que se enmarca este proyecto está implementada en este lenguaje C++, por lo que por extensión, se implementa la capa de red en este mismo lenguaje. Para el desarrollo del código se ha utilizado el entorno de desarrollo integrado de Eclipse en su versión Indigo, con el plugin para integrar Subversion y poder controlar las versiones.

4.2. Detalles de la implementación

Partiendo del código fuente de la plataforma de Javier Celaya, se ha implementado todo el protocolo de la capa de red en forma de árbol. El código se ha desarrollado con una idea global, en la que posteriormente iba a haber dos escenarios de comprobación: Simulación y Real.

Para el escenario real se implementa la DHT mediante el proyecto Apache Cassandra. También se modifica el código, para que el programa reciba por parámetro a través de qué nodo se va a insertar el nuevo nodo. Si este parámetro no está presente, la dirección local será la elegida por defecto.

Para el escenario de simulación, el simulador reemplaza las funciones de comunicación y de paso de mensajes de red para simularlas. Asimismo, se sobrescriben los métodos de la DHT, y se implementa en memoria una tabla hash mediante la clase map, que es un contenedor que guarda elementos formados con la combinación del par Clave-Valor.

La simulación continúa ejecutándose hasta que se cumple una condición de parada, en la que se tiene que cumplir que todos los nodos estén conectados en la red y estén equilibrados, momento en el que la simulación finaliza. Además, tras realizar las simulaciones, hay que comprobar el resultado obtenido, por lo que se implementan métodos de comprobación de la condición de parada, comprobación del estado final de todos los nodos y de representación de la estructura final.

4.3. Simulador

Para comprobar el funcionamiento de la capa de red se ha utilizado un simulador de eventos discretos. Ésta técnica de modelado dinámico de sistemas se caracteriza por un control de la variable tiempo, en la que se avanza a intervalos variables por medio de eventos.

El simulador utilizado ha sido desarrollado por Javier Celaya para validar la plataforma de tareas distribuidas que está desarrollando. Este simulador trata de minimizar el uso de la memoria, permitiendo maximizar el número de nodos que se puedan simular en la red. A este simulador se le han añadido funciones que permiten representar la estructura del árbol y toda la información de los nodos, así como funciones que permiten comprobar el correcto estado de la estructura. Además, está desarrollado en el lenguaje C++, por lo que se adapta perfectamente a la capa de red que se implementa en el mismo lenguaje.

4.4. Medidas del código fuente

El código fuente desarrollado se puede dividir en dos grupos.

a) **Capa de red**
Se han escrito 24 clases, con un total de 7189 líneas de código.
- 7 clases pertenecientes al núcleo de la estructura, entre las que destacan la DHT y las clases de los dos tipos de nodos, con un total de 6366 líneas de código.
- 17 clases, para todos los mensajes de intercambio entre los nodos, con un total de 823 líneas de código.

b) **Simulador**
Se han utilizado 5 clases, con un total de 1113 líneas de código.
- Se modifican 3 clases ya existentes para adaptarlas a la nueva capa de red, con un total de 410 líneas de código.
- Se ha crea 1 clase, donde se sobrescriben los métodos de la DHT en memoria, con un total de 35 líneas.
- Se ha crea 1 clase, donde se definen los casos de simulación, con un total de 668 líneas de código.

Todo ello hace un total de 8302 líneas de código.

5. Simulación y prueba real

En el primer apartado, se comenta todo lo realizado en cuanto a simulación. Se crea cómo se crea el escenario, es decir, los procesos de inserción y simulación de fallos. Posteriormente, se trata cómo se comprueba si el resultado es correcto y finalmente la batería de pruebas y resultados. En el segundo apartado se comenta la prueba en un escenario real.

5.1. Pruebas con el simulador

5.1.1. Escenario

En todas las simulaciones, en primer lugar hay que crear la red. Para ello se introducen de forma secuencial diez nodos físicos. Estos nodos son los nodos públicos bien conocidos, que actuarán como punto de entrada a la red. Cualquier nodo que quiera formar parte de la red tiene que realizar la petición de inserción a uno de estos nodos.

Una vez que la red está estabilizada, los intervalos de inserción de los nuevos nodos físicos se generan aleatoriamente con un proceso de Poisson de media 1 segundo. En cada nueva inserción de un nodo físico, el punto de entrada se selecciona aleatoriamente de manera uniforme entre los diez ya existentes. Este proceso se repite hasta introducir el número de nodos que se quieren simular, que viene en el fichero de configuración del simulador.

Por otra parte, el fichero de configuración del simulador también nos permite configurar los fallos. Se programa con un proceso de Poisson de media configurable, qué nodo aleatorio es el que falla. Posteriormente, con un intervalo de tiempo mucho mayor que el tiempo de detección de fallo, el nodo solicita la reentrada a la red. Al reentrar será como uno nuevo, no guarda ninguna información de su presencia anterior.

5.1.2. Comprobación del resultado

Para comprobar el resultado tras la simulación se han realizado pruebas de validación. Estas pruebas son el proceso de revisión en el que se comprueba si el resultado obtenido cumple o no los requisitos especificados, es decir, nos indica si el resultado es correcto o no. Al finalizar la simulación se comprueban los siguientes parámetros:

- Todos los nodos tienen la misma raíz del árbol. Esto comprueba que no haya ramas del árbol independientes.
- Todos los nodos con rol de estructura están en el estado ONLINE y su identificador de transacción es nulo. Esto nos indica que el nodo no está en mitad de ninguna transacción.
- Todos los nodos con rol de estructura están equilibrados, es decir, la diferencia de la altura de los subárboles izquierdo y derecho es máximo una unidad. Esto comprueba que el árbol esta balanceado.
- Se recorren todos los nodos del árbol en preorden y el número de nodos recorridos tiene que coincidir con el doble de nodos físicos menos uno (Por la propiedad comentada en la sección 3.1).

Si todo lo anterior se cumple, el resultado obtenido es correcto, de lo contrario algo falla.

5.1.3. Validaciones básicas

Presentado el escenario y la forma de comprobar el resultado se realizan diversas pruebas. Siendo N el valor que representa al número de nodos físicos que van a formar parte de la red, N toma los siguientes valores: 10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000 y 500000.

En primer lugar se simula la red para los valores de N definidos anteriormente sin ningún nodo que vaya a fallar. Para todos los valores el resultado obtenido basándonos en las comprobaciones anteriores es el correcto. Por lo tanto la escalabilidad se ha conseguido.

Posteriormente se simula la red para los valores de N definidos anteriormente y programando el fallo de un nodo aleatorio en un tiempo aleatorio. Para todos los valores la red ha detectado el nodo que ha fallado, se ha reconstruido la estructura, y posteriormente se ha vuelto a introducir el nodo que había fallado. A continuación, se realizan las mismas pruebas pero esta vez programando varios fallos durante la simulación. Siempre y cuando un nodo que ha fallado no esté involucrado en la reconstrucción de otro nodo, la red consigue detectar todos los fallos, reconstruir la estructura y volver a introducir los nodos en la red.

Sin embargo, existe una limitación en el protocolo, que se produce si uno o más nodos caídos están involucrados en un proceso de reconstrucción de otro nodo, en la que la recuperación de la estructura no puede llevarse a cabo. Esto se produce porque el coordinador de la reconstrucción envía mensajes a nodos que han fallado y permanece a la espera de las respuestas de estos, que nunca van a llegar por lo que todos los nodos involucrados quedan bloqueados.

Por lo que se puede asegurar que la estructura se recupera correctamente tras un fallo, siempre y cuando no haya ningún nodo caído involucrado en la reconstrucción de otro nodo.

5.1.4. Validación en caso de churn

Las redes P2P tienen un gran dinamismo, es decir, que la red está cambiando continuamente debido a las conexiones y desconexiones de los participantes. El proceso que involucra la entrada y salida de los participantes de la red de manera arbitraria se denomina churn.

En este apartado, se profundiza en la tolerancia a fallos sometiendo a la red P2P a un permanente cambio. Se programan inserciones de nodos exactamente igual que en el apartado anterior, y a su vez se programan continuamente fallos aleatorios de nodos físicos con un proceso de Poisson de media 10 segundos. Por la limitación de este proyecto, que sólo permite reconstruir fallos en los que no intervengan otros nodos caídos, en el momento de programar el fallo de un nodo físico se comprueba que todos los nodos vecinos estén dentro de la red y no se haya programado ningún fallo para esos nodos. Con esta configuración, se ha conseguido crear un escenario en el cual hay continuas entradas y salidas de participantes de la red.

Durante un tiempo considerable de simulación, la capa de red se comporta perfectamente detectando los fallos, y posteriormente, procediendo a la reconstrucción devolviendo la red a un estado correcto. Con lo que se puede afirmar que la red responde bien frente a un churn elevado.

Sin embargo, debido a la limitación a la hora de reconstruir nodos con otros nodos caídos, eventualmente aparecen casos de error más complejos. Más concretamente en nuestro caso, se produce porque en el momento de programar el fallo de un nodo todos sus vecinos están en la red, sin embargo, en el momento que se inicia la reconstrucción alguno está caído y aún no se ha realizado su reconstrucción. En este caso, todos los nodos quedan bloqueados imposibilitando el correcto funcionamiento de la red. Un ejemplo de la situación descrita es la que podemos ver en la figura 5.1, donde se pueden observar todos los nodos que se usarían en una reconstrucción del nodo 92 y 38 respectivamente.

Si se programa el fallo del nodo físico 92 y posteriormente del nodo físico 38 y las reconstrucciones se realizan en este orden no hay ningn problema. Sin embargo, si por algún motivo se inicializa la reconstrucción del nodo 38 en primer lugar la reconstrucción no

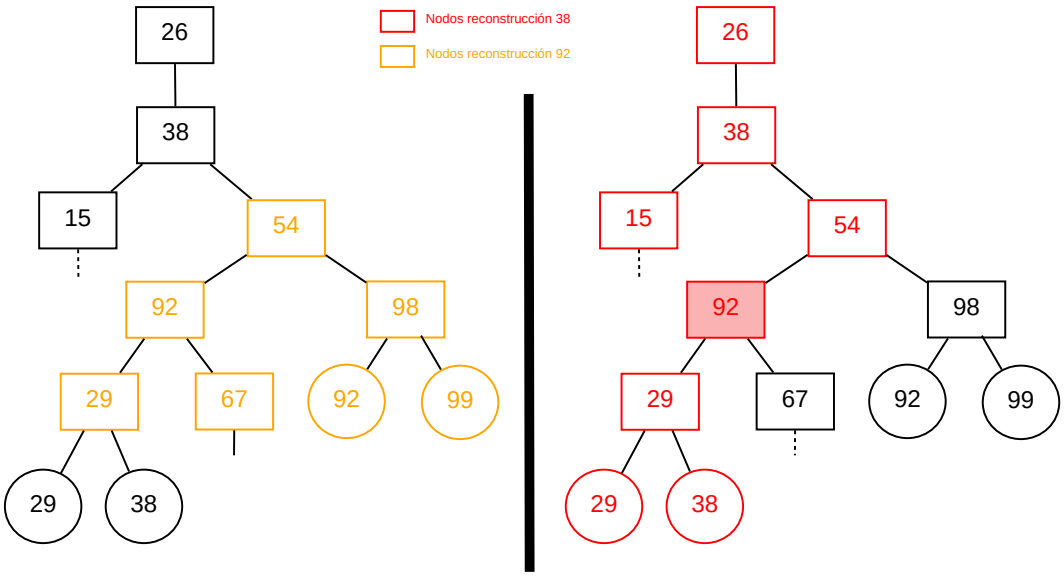

Figura 5.1: Misma situación resaltando los nodos involucrados en la reconstrucción del nodo 92 y 38 de manera independiente.

puede finalizar por involucrar al nodo 92 quedando bloqueados todos los nodos coloreados en la figura 5.1. Es por ello que esta limitación en el funcionamiento se solucionará en el futuro.

5.1.5. Consumo de Ancho de Banda

Además de la validación se ha realizado una medida del consumo de ancho de banda de salida y de entrada de los nodos de la red. Se han obtenido dos valores: la media de uso de ancho de banda de todos los nodos y el pico de máximo uso de ancho de banda en periodos de diez segundos. En las figuras 5.2 y 5.3 se pueden observar las medidas obtenidas para el ancho de banda de salida y de entrada respectivamente.

Para realizar las pruebas se realizan varias simulaciones. Siendo N el valor que representa al número de nodos que van a formar parte de la red, N toma los siguientes valores: 10, 100, 1000, 10000, 100000. Además para cada valor de N se han realizado siete simulaciones distintas. De los siete resultados obtenidos se eliminan las muestras más extremas y se realiza la media de las restantes.

Las líneas azules representan el máximo uso de ancho de banda en un período de diez segundos y las líneas rojas describen la media de uso de ancho de banda de todos los nodos.

Hay que destacar que el máximo uso del ancho de banda de salida se situa en tan solo el 3,32 KBps y el máximo uso del ancho de banda de entrada es solamente el 1,09 KBps. Por otra parte la media de uso de ancho de banda de entrada y de salida se sitúa

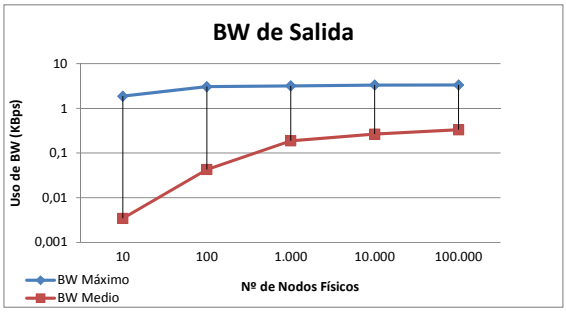

Figura 5.2: Medidas del uso del ancho de banda de salida respecto al número de nodos de la red.

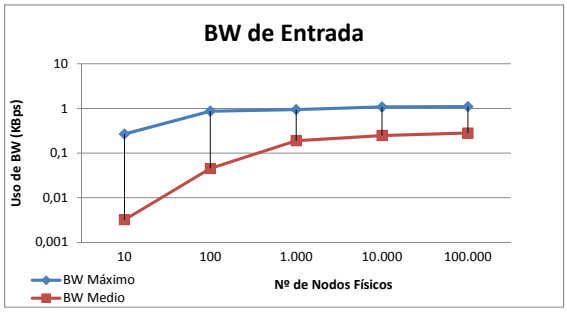

Figura 5.3: Medidas del uso del ancho de banda de entrada respecto al número de nodos de la red.

únicamente en 0,31 KBps. Ante los resultados obtenidos en ambos casos hay que concluir que frente a un crecimiento exponencial del tamaño de la red el aumento del uso del ancho de banda es practicamente despreciable.

5.1.6. Tiempo medio de inserción

También se ha realizado una medida del tiempo de inserción de los nodos en la red. Para realizar las pruebas se realizan varias simulaciones. Siendo N el valor que representa al número de nodos que van a formar parte de la red, N toma los siguientes valores: 10, 100, 1000, 10000, 100000. Además para cada valor de N se han realizado siete simulaciones distintas. Se obtiene el tiempo que tarda en insertarse en la red el último nodo. El tiempo se determina desde que dicho nodo envía el mensaje de inserción InsertCommandMsg hasta que los dos roles que interpreta cada nodo realizan el commit y pasan a formar parte de la red. De los siete resultados obtenidos se eliminan las muestras más extremas y se realiza la media de las restantes.

En la figura 5.4, representado con la línea azul, se puede ver el tiempo de inserción del último nodo de la red. Como se puede observar, frente a un crecimiento exponencial del tamaño de la red el aumento del tiempo de inserción es mínimo, situándose el tiempo medio de inserción entorno 2,25 segundos para redes de 100.000 nodos.

Lo anteriormente expuesto es otra evidencia de que la capa de red es escalable en cuanto al número de participantes en la misma, ya que atendiendo a la gráfica para redes más grandes, no afecta al rendimiento de la misma.

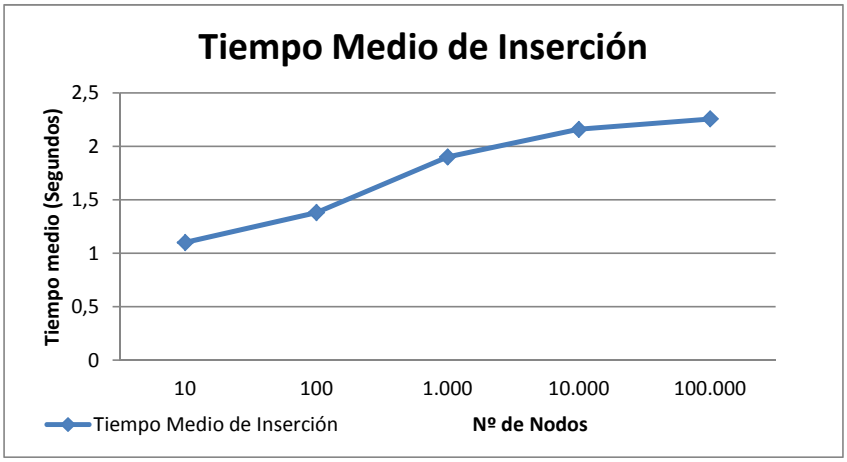

Figura 5.4: Medida del tiempo de inserción del último nodo que accede a la red respecto del número de nodos de la red

5.2. Prueba escenario real

Esta prueba trata de validar y comprobar el correcto funcionamiento de la capa de red en un escenario real. Para ello se usan 6 ordenadores del laboratorio 1.3b. En cada uno de ellos se lanzarán 3 instancias de la capa de red, por lo que se creará un red de un total de 18 nodos físicos.

A la hora de crear la red, cada instancia tiene que recibir dos parámetros el puerto propio donde se ejecuta (-p Número) y el punto de entrada a través del cual se va a conectar a la red (-e IP:Puerto). Salvo la excepción del primer nodo que solo necesita el parámetro del puerto. Recordando que el identificador de cada nodo es el par dirección IP - Puerto, hay que destacar que cada instancia en un mismo ordenador es obligatorio que tenga un puerto distinto.

Tras lanzar los 18 nodos físicos, se comprueba que se han introducido correctamente todos los nodos en la red. A continuación, para comprobar la tolerancia a fallos se detiene la ejecución de un nodo. En el momento que se detecta el fallo, se inicia el proceso de reconstrucción de la red y tras completarse dicho proceso, la estructura continúa ejecutándose correctamente.

De esta forma se ha realizado la prueba en una escenario real y se puede asegurar que ha sido un éxito, ya que se ha creado la red y se ha reconstruido tras un fallo.

6. Conclusiones

En este proyecto fin de carrera se ha conseguido diseñar e implementar una capa de red jerárquica, escalable y tolerante a fallos. Se ha conseguido obtener una estructura totalmente distribuida, sin memoria compartida y creando un protocolo de intercambio de mensajes. Se ha validado el correcto funcionamiento de la capa de red a través de un simulador, y posteriormente en una prueba real en el laboratorio. En la primera parte de simulación, se ha comprobado que la capa de red se adapta a cualquier tamaño de red. Se han realizado pruebas de simulación de la red desde un único nodo físico, hasta 500.000 nodos físicos. En toda la batería de pruebas realizadas, el resultado ha sido el correcto. Además, las medidas de tiempo de inserción y la medida del uso del ancho de banda de entrada y salida han arrojado resultados muy adecuados a lo que se buscaba. Por este motivo podemos asegurar que se ha conseguido el objetivo buscado de la escalabilidad.

En las pruebas se han introducido fallos de nodos físicos aleatorios, y en todas ellas la red ha detectado dicho fallo. Posteriormente se inicia el proceso de recuperación, teniendo en cuenta que en este proceso no esté involucrado otro nodo caído, y se reconstruyen los enlaces rotos devolviendo la estructura a un estado correcto. El resto de nodos no involucrados no se ven afectados y siguen funcionando correctamente. Con estos datos se concluye que se ha conseguido el objetivo de tolerancia a fallos para las situaciones más corrientes.

Se ha desarrollado una capa de red que servirá como medio para la distribución de la información y la localización de recursos a emplear por otras plataformas. Todo lo expuesto demuestra que todos los objetivos de este proyecto se han cumplido.

6.1. Trabajo futuro

Una vez que se ha visto que las decisiones tomadas han sido acertadas, el siguiente paso es profundizar en la tolerancia a fallos. Se debe estudiar de forma más detallada el tratamiento y recuperación de errores de varios nodos próximos entre sí. Es decir, nodos caídos que intervienen en el proceso de reconstrucción de otro nodo caído.

Otra mejora futura es la optimización del protocolo de intercambio de mensajes para

reducir el tráfico de mensajes. Por un lado, agrupar varios mensajes dirigidos a un mismo destino desde una misma fuente, y por otra parte, mejorar el protocolo de detección de nodos fallidos y de esta forma conseguir una reconstrucción en el menor tiempo posible.

Con el objetivo de evitar usuarios malintencionados habría que implementar mecanismos de seguridad en la DHT para proteger los datos y que sólo el propietario pueda modificarlos.

6.2. Valoración Personal

Este ha sido sin duda el trabajo más importante y difícil al que me he tenido que enfrentar. Afrontar los problemas en el desarrollo de un proyecto de esta envergadura, me ha aportado numerosos conocimientos técnicos que serán de mucha utilidad durante el desarrollo de mi vida profesional. He aprendido las pautas a seguir en la realización de un proyecto de investigación, desde la fase de recopilación de información y búsqueda de artículos similares hasta la obtención de las conclusiones, así como el análisis del problema, el diseño de una solución y la realización de las pruebas para comprobar la validez del resultado.

Este proyecto me ha permitido profundizar mucho en el lenguaje orientado a objetos C++. Por un lado al escribir numerosas líneas de código desde cero, y por otro al leer y comprender código fuente desarrollado por otros programadores para adaptarlo e integrarlo todo en un mismo proyecto. También me ha permitido poder adentrarme en el paradigma de la programación distribuida, conocer y posteriormente poder dar una solución a los numerosos problemas que ella entraña. Considero que la programación distribuida es una tecnología muy extendida por el auge que han tenido las redes P2P, y conocerla puede serme de gran utilidad.

A modo personal, este proyecto ha hecho que desarrolle una visión global que me permita anticipar los inconvenientes futuros que siempre aparecen en el transcurso de un proyecto real de desarrollo de software, y de esta forma poder resolver estos problemas de manera más eficiente. Asímismo ha sido enriquecedor el hecho de haber conseguido los objetivos marcados al principio, y por ello quiero agredecer a mi director todo el tiempo y dedicación que me ha brindado a lo largo de este tiempo.

Bibliografía

[1] The apache cassandra project. http://cassandra.apache.org/.

[2] Avl tree. http://xlinux.nist.gov/dads//HTML/avltree.html.

[3] Maidsafe. kademlia dht with nat traversal. http://code.google.com/p/maidsafe-dht/.

[4] Apache. The apache project. http://www.apache.org/.

[5] Apache. Libqtcassandra. http://snapwebsites.org/project/libqtcassandra.

[6] J. Celaya. A highly scalable decentralized scheduler of tasks with deadlines. *Grid Computing (GRID), 2011 12th IEEE/ACM International Conference*, Septiembre 2011.

[7] C.S. Ellis. Concurrent search and insertion in avl trees. *Computers, IEEE Transactions*, C-29(Issue: 9):811 – 817, Sept 1980.

[8] H.V. Jagadish, B.C. Ooi, and Q.H. Vu. Baton: a balanced tree structure for peer-to-peer networks. *VLDB '05*, 2005.

[9] H.V. Jagadish, B.C. Ooi, Q.H. Vu, R. Zhang, and A. Zhou. Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. *ICDE '06.*, Abril 2006.

[10] A. Kaluszka. Distributed hash tables. http://courses.ischool.berkeley.edu/i250/s10/report/, April 2010.

[11] B.W. Lampson. Atomic transactions. *Distributed Systems:Architecture and Implementation*, Vol. 105:246–265, 1981.

[12] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. *In Peer-to-Peer Systems*, vol. 2429(Lecture Notes in Computer Science):53–65, Springer Berlin / Heidelberg 2002.

[13] C. Mohan and B. Lindsay. Efficient commit protocols for the tree of processes model of distributed transactions. *ACM SIGOPS Operating Systems Review*, Volume 19(Issue 2):40 – 52, April 1985.

[14] O. NURMI, E. SOISALON-SOININEN, and D. WOOD. Concurrent balancing and updating of avl trees. *http://www.csa.com*, 1992.

[15] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. *SIG-COMM Comput. Commun. Rev.,31(4)*, pages 161–172, August 2001.

[16] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, Springer-Verlag 2001.

[17] F. Ruiz and M.A. Moraga de la Rubia. El modelo de datos jerarquicos. *Universidad de Castilla-La Mancha*, Abril 2001.

[18] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *SIGCOMM 01*, pages 149–160, August 2001.

[19] R Van Renesse, K.P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[20] P. Yalagandula and M. Dahlin. A scalable distributed information management system. ACM, 2004.

[21] B.Y. Zhao, J.D. Kubiatowicz, and A.D Joseph. Tapestry: An infrastructure for fault tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141. UC Berkeley, April 2001.