PROYECTO FIN DE MÁSTER

POSGRADO EN INFORMÁTICA

UNIVERSIDAD DE ALMERIA

ESCUELA POLITÉCNICA SUPERIOR

MÁSTER EN INFORMÁTICA AVANZADA E INDUSTRIAL

GESTIÓN DE LA INFORMACIÓN ESPACIAL EN SISTEMAS P2P

Curso 2012/2013

Alumno/a:
Jesús Vallecillos Ruiz

Director/es:
Antonio Leopoldo Corral Liria
Luis Fernando Iribarne Martínez

UNIVERSIDAD DE ALMERÍA
ESCUELA POLITÉCNICA SUPERIOR
Departamento de Lenguajes y Computación

TRABAJO FIN DE MÁSTER
MÁSTER EN INFORMÁTICA AVANZADA E INDUSTRIAL
POSGRADO EN INFORMÁTICA

GESTIÓN DE LA INFORMACIÓN ESPACIAL EN SISTEMAS P2P

Jesús Vallecillos Ruiz

Dirigida por:
Dr. Antonio Leopoldo Corral Liria
Dr. Luis Fernando Iribarne Martínez

Almería, Septiembre 2013

TRABAJO FIN DE MÁSTER
MÁSTER EN INFORMÁTICA AVANZADA E INDUSTRIAL
POSGRADO EN INFORMÁTICA

GESTIÓN DE LA INFORMACIÓN ESPACIAL EN SISTEMAS P2P

por
Jesús Vallecillos Ruiz

Para la obtención del
Título del Máster en Informática Avanzada e Industrial
Posgrado en Informática

Director	**Director**	**Autor**
Dr. Antonio Leopoldo Corral Liria	Dr. Luis Fernando Iribarne Martínez	Jesús Vallecillos Ruiz

A mi familia por haber invertido en mi

tanto trabajo y esfuerzo para formarme como persona.

A mis directores Antonio Corral y Luis Iribarne

por su inversión y tiempo para ayudarme a mejorar.

A Juan Álvaro Muñoz por haberme facilitado y explicado

documentación relacionada con el trabajo.

Índice

Gestión de la información espacial en sistemas P2P

Jesús Vallecillos Ruiz

***Abstract*—Nowadays, the spatial data management is useful in many application areas such as geoscience, CAD, robotics and environmental protection, just to cite a few. Dealing with such data (points, lines, polygons, regions, etc..) can be very costly. Therefore, spatial indexing methods (IE) emerged with the aim of improving the efficiency of this type of query data. With respect to Peer-to-Peer systems (P2P), this is an alternative to distributed systems because they are dynamic networks that aim to share resources in a distributed manner. The resources range from multimedia files, data, processing cycles, etc.. This system consists of pairs which provide the network get benefits like: *autonomy*, because the pairs do not follow any rules about the way in which they should behave in the system and how they must share the resources; *symmetry* as pairs can act as clients, making use of other resources that pairs share. In this paper we try to make use of the advantages of P2P systems, for the application to spatial information. For this, as it is drawn along the document, it must be performed a pretreatment to integrate spatial information within such networks, from the information of a 2D to 1D space using some of the methods listed below. In addition, we can observe the difference when working with different protocols for the network in which we have integrated spatial information. To carry out the study, a review of the tools has been previously done to simulate P2P systems, of which we have selected *PeerSim* to facilitate the reuse of already developed protocols through libraries. In this simulator we have worked with P2P protocols such as *Pastry*, *Kademlia* and *Chord* on which we have simulated the spatial information access of raster and vector type. To cite an example, in one of the experiments conducted among protocols, we have measured the difference between increasing the number of pairs that manage vector and raster information, to find a point in space in order to observe the number of hops that took place on the network to search locate information.**

***Resumen*—La gestión de datos espaciales es útil hoy en día en muchos campos de aplicación, tales como la geociencia, CAD, la robótica o la protección del medio ambiente por mencionar algunos. Tratar con este tipo de datos (puntos, líneas, polígonos, regiones, etc.) puede llegar a ser muy costoso. Por ello, surgieron los métodos de Indexación Espacial (IE) con el objetivo de mejorar la eficiencia de las consultas en este tipo de datos. Con respecto a los sistemas Peer-to-Peer (P2P), estamos ante una alternativa a los sistemas distribuidos ya que son redes dinámicas que tienen la finalidad de compartir recursos de forma distribuida. Los recursos van desde archivos multimedia, datos, ciclos de procesamiento, etc. Este sistema está formado por *pares* los cuales consiguen dotar a la red de ventajas como: *autonomía*, ya que los pares no siguen regla alguna sobre la manera con la que deben de comportarse en el sistema y como deben de compartir los recursos; *simetría*, pues los pares pueden actuar como clientes, haciendo uso de los recursos que otros pares comparten. Con este trabajo pretendemos hacer uso de las ventajas de los sistemas P2P, para aplicarlos a información espacial. Para ello, como se redacta a lo largo del documento, hay que realizar un tratamiento previo de la información espacial para poder integrarla dentro de este tipo de redes, pasando dicha información de un espacio de 2D a 1D utilizando algunos de los métodos que aparecen a continuación. Además, podremos comprobar la diferencia a la hora de trabajar con diferentes protocolos P2P para la red en la que hemos integrado la información espacial. Para llevar a cabo nuestro estudio, previamente se ha realizado una revisión de las herramientas que permiten simular los sistemas P2P, de las cuales hemos seleccionado *PeerSim* por facilitar la reutilización de protocolos ya desarrollados por medio de librerías y estar desarrollado en *Java*. En dicho simulador se ha trabajo con protocolos P2P como *Pastry*, *Kademlia* y *Chord* sobre los que hemos simulado el acceso a información espacial de tipo ráster y vectorial. Por citar un ejemplo, en uno de los experimentos realizados entre los protocolos, hemos medido la diferencia que existe entre aumentar el número de pares que gestionan la información vectorial y ráster, para buscar un punto en el espacio con el objetivo de observar el número de saltos que tuvo lugar en la red para localizar la información.**

Keywords*—*spatial Information, P2P systems , P2P protocols, simulation, experimental results.

Palabras clave*—*Información espacial, sistemas P2P, protocolos P2P, simulación, resultados experimentales.

I. Introducción

EL paradigma de red de *pares* (a lo largo del documento un *par* también podrá referirse con el término *nodo*) o red punto a punto (P2P del ingles Peer-to-Peer), ha llegado a ser muy popular para almacenar y compartir información de una manera totalmente descentralizada. Se trata de una red de computadoras en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de pares que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás pares de la red. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados manteniendo links (índices) a otros pares. Los sistemas P2P

- proporcionan un método para distribuir la información disponible a los pares,
- garantizan la recuperación de alguna información que existe en el sistema,
- logran un tamaño de índice razonable para los pares,
- localizan un camino de búsqueda razonable para realizar alguna búsqueda en el sistema,
- mantienen un bajo coste para actualizar los índices de los pares cuando los pares se acoplan a la red o se marchan,
- localizan datos y buscan llevar a cabo un balanceo de la carga, por ejemplo, para aquellos pares que no están sobrecargados con datos y para aquellas zonas de la red en las que se produce un embotellamiento.

Hasta hace poco, las búsquedas se han centralizado mayoritariamente en sistemas P2P que manejan datos en una dimensión (1D) tales como string y números. Sin embargo, ha surgido la necesidad de gestionar datos multidimensionales tales como son los datos espaciales en aplicaciones P2P. Estos

sistemas poseen requerimientos adicionales que extienden las particularidades de los datos. En aplicaciones multidimensionales centralizadas, la información se almacena acorde a su extensión multidimensional usando una estructura de índice (por ejemplo, R-tree [Guttman, 1984]). Normalmente, estas estructuras conservan la *localización* y el *direccionamiento* de información espacial. Intuitivamente, la localización implica que información espacial próxima se almacena en pares cercanos, mientras la direccionalidad implica que estructura de índices conservan la orientación. Las nociones de localidad y direccionalidad son muy importantes. Si una estructura de índices conserva estas propiedades entonces la búsqueda se puede mejorar mucho en cuanto a coste de la evaluación de la consulta [Rigaux et al., 2001].

La gestión de datos espaciales es útil hoy en día en muchos campos de aplicación, tales como la geociencia, CAD, la robótica o la protección del medio ambiente por mencionar algunos. Aunque tratar con este tipo de datos conlleva tener que resolver ciertos problemas relacionados por ejemplo con las consultas. Por ello existen los métodos de indexación espacial (IE) con el objetivo de mejorar la eficiencia de la consulta en este tipo de datos. Actualmente, el paradigma de consulta en la IE ha pasado desde enfoques centralizados a descentralizado, haciendo uso de redes P2P gestionadas por métodos de IE. Desde mediados del siglo pasado hasta hoy, la IE se ha desarrollado a gran velocidad. La idea principal de la IE es organizar los datos desde una vista global, donde toda la dimensionalidad se tiene en cuenta de forma sintetizada. El proceso de consulta se acelera llevando a cabo mejoras en la estructura de indexación y en los algoritmos de búsqueda. La ventaja importante que introduce la IE es que permite realizar un proceso de poda sobre muchas zonas de búsqueda del espacio que son inútiles en el proceso de consulta.

Como soporte para los datos espaciales (como por ejemplo puntos, segmentos de línea, regiones, etc.), los sistemas de redes P2P [Lua et al., 2005] surgen como un nuevo paradigma en las últimas décadas. En estos sistemas los *pares* de computación que forman el sistema son independientes y autónomos, y ellos cooperativamente logran la computación y forman un auto-organismo. Por lo que, las redes P2P consiguen proporcionar un método para distribuir la información disponible entre los pares, garantizando la recuperación de información que existe en el sistema, logrando un camino de búsqueda razonable, manteniendo un coste bajo en la actualización de los índices cuando un par se une o se marcha, y realizando balanceo de búsqueda y carga de datos en los pares cuando haya pares que estén sobrecargados por alguna de estas operaciones. Estos sistemas dan uno de las mejores rendimientos, producidos por los cuellos de botella que tienen lugar en los sistemas centralizados. Además los sistemas P2P, para poder tratar con datos multidimensionales, necesitan ser equipados con capacidad de procesamiento de consultas complejas multidimensionales [Cai et al., 2004] [Tanin et al., 2007]. En los entornos P2P, la mayoría de usuarios solicitan consultas complejas para encontrar objetos que coincidan con requerimientos multidimensionales. Por ejemplo, en los juegos multi-jugador P2P [Lee et al., 2005], redes de búsqueda de trabajo P2P [Tanin et al., 2007] o subastas P2P, la gente normalmente quiere encontrar otros jugadores en un área específica (geografía 2-dimensiones o una en 3-dimensiones), por lo que para resolver estos trabajos son necesarios requerimientos multidimensionales. Las técnicas P2P tradicionales pueden proporcionar capacidades de consultas exactas pero mecanismos de poca calidad para búsqueda en rangos multidimensionales y búsquedas por similitud. Por eso, los sistemas de redes P2P deben ser equipados con componentes de IE.

En la literatura, existen dos paradigmas diferentes para manejar datos multidimensionales en sistemas P2P. El primer paradigma propone el uso de una versión distribuida de algún índice multidimensional centralizado [Jagadish et al., 2005], [Jagadish et al., 2006], [Mondal et al., 2005], [Tanin et al., 2007]. El principal reto de esta propuesta es evitar los cuellos de botella que tienen lugar en la raíz del árbol (cada búsqueda debe pasar a través de este par). En [Tanin et al., 2007] se observa como se alivia la carga de tráfico de la raíz iniciando la búsqueda en un nivel prefijado inferior a la raíz. Por otro lado, el trabajo en [Tanin et al., 2007] se proponen índices que ayuden a evitar la búsqueda en el índice de la raíz.

El segundo paradigma mapea los datos multidimensionales en una dimensión única y usa una Tabla de Dispersión Distribuida (TDD) [Cai et al., 2004], [Ganesan et al., 2004], [Lee et al., 2005], [Sahin et al., 2005]. Brevemente, las técnicas para TDD de 1D usan una distancia métrica para definir la localización de los datos en 1D. Entonces, los pares usan la distancia para almacenar datos e indexar otros pares. Normalmente, los pares almacenan datos e indexan pares acorde a su posición. Además, la búsqueda se desarrolla basada en la distancia mas próxima y en la información indexada disponible en cada par. Hay requerimientos similares para manejar datos multidimensionales en sistemas P2P. Hay una necesidad de un almacenamiento, una indexación, y un framework de consulta que pueda saber la localización de forma directa de cada uno de los espacios multidimensionales. Existen trabajos ([Cai et al., 2004], [Lee et al., 2005], y [Sahin et al., 2005]) para uso de datos multidimensionales usando una curva de llenado de espacio (Z-order) y a partir de ahí definir una distancia acorde a esta ordenación. El reto de dicha propuesta es mantener las propiedades importantes del espacio (localización y direccionalidad).

La principal motivación de este trabajo ha sido la de intentar gestionar información espacial tanto de tipo ráster como de tipo vectorial dentro de sistemas P2P, con el objetivo de aprovechar la potencia de este tipo de sistemas con información que requiere de complejidad de cómputo para su manejo. Para ello, se han hecho uso de protocolos P2P simulados que han sido adaptados para poder trabajar con este tipo de información.

El documento que expone el trabajo realizado ha sido divido en las siguientes secciones, en la Sección II hacemos una introducción a las redes P2P explicando sus propiedades. En la Sección III hablamos de los tipos de sistemas P2P con los que nos podemos encontrar, que están divididos básicamente en estructurados y no estructurados. A continuación, en la Sección IV exponemos los protocolos más extendidos y más comunes dentro de los sistemas P2P. Luego detallamos el concepto de

información espacial en la Sección V. Para tener una forma de trabajar con la información espacial tenemos la Sección VI donde se habla de la indexación espacial basada en P2P. Hacemos un recorrido de las herramientas de estudio actuales para los sistemas P2P en la Sección VII, donde podemos ver algunos de los simuladores que nos permiten trabajar con redes P2P. En la Sección VIII, exponemos la implementación de los protocolos P2P que hemos utilizado para el desarrollo del trabajo, todos ellos implementados sobre el simulador *PeerSim*. Después tenemos la Sección IX donde vemos los resultados obtenidos en las simulaciones realizadas y por último la Sección X, donde se pueden ver las conclusiones obtenidas tras el trabajo realizado junto con futuras tareas a realizar para continuar con el estudio en este campo.

II. Sistemas de redes P2P

A continuación, vamos a detallar las propiedades de los sistemas P2P, para ello se ha realizado un resumen procedente de [Pérez-Miguel et al., 2009]. Los sistemas P2P surgen como una alternativa en arquitectura de los sistemas distribuidos. Se definen como redes dinámicas que tienen la finalidad de compartir recursos de forma distribuida. Estos recursos pueden ir desde archivos multimedia, datos, memoria hasta ciclos de procesamiento. La red está integrada por pares que poseen características especiales:

- **Autonomía**: Los pares no siguen regla alguna sobre la manera con la que deben actuar con el sistema ya que, en principio, el sistema no impone restricciones sobre cuántos y de qué forma se deben compartir los recursos. La autonomía de los pares les permite decidir su tiempo de permanencia en el sistema, ésto implica que la composición y cantidad de los pares es variable.
- **Simetría**: Los pares pueden actuar como clientes, solicitando y haciendo uso de los recursos que otros pares comparten; o servidores, compartiendo los recursos que proporcionan al sistema.

Debido a los puntos anteriores, los sistemas P2P obtienen características para organizar a los pares y respetar las propiedades de éstos. Las características que se enuncian a continuación son generales y es probable que un sistema P2P específico no las posea todas:

- **Descentralización**: Los recursos de interés se encuentran distribuidos entre los pares, por tal motivo se usan pocos o ningún servicio centralizado. Si se hace uso de alguna entidad centralizada, ésta solo es empleada como un servidor que gestiona las consultas de los pares, sin embargo, el intercambio de recursos se hace directamente entre los pares.
- **Autoorganización**: El sistema realiza de forma automática las interconexiones posibles para integrar a los pares a la red y darles el acceso a los recursos.
- **Balanceo de carga**: El flujo de datos se reparte naturalmente entre los pares que se encuentran en el sistema ya que los recursos y la mayor parte de la funcionalidad del sistema residen de forma distribuida en ellos.
- **Espacio de nombres**: Es común que los pares se unan al sistema con una dirección IP diferente en cada ocasión, por lo que se hace uso de un espacio de nombres superpuesto al de Internet para localizar tanto a los pares como a los recursos por un identificador y no por su dirección IP.
- **Escalabilidad**: Se superan los problemas de escalabilidad inherentes al modelo cliente-servidor ya que el sistema no depende de una entidad centralizada para otorgar los recursos.

Por tanto, desde el surgimiento de las redes P2P se ha debatido mucho sobre la posibilidad de usar este paradigma de sistema distribuido como base para montar un sistema de computación [Foster and Iamnitchi, 2003]. Este sistema se puede definir como un conjunto de computadores interconectados entre si por una red de comunicaciones que intentan unir sus recursos para llevar a cabo algún tipo de tarea computacional. Cada computador conectado al sistema posee sus propios recursos computacionales independientes, sin embargo, desde el punto de vista del usuario, el sistema se percibe como un único sistema. En dicho sistema un usuario accedería a los recursos de la misma forma en que accede a sus recursos locales. Sin embargo, el hecho de que el sistema esté repartido entre múltiples entidades le da unas características en cuanto a escalabilidad y soporte a fallos que muy pocos poseen.

Sistemás que cumplan estas condiciones hay muchos, desde clústeres hasta sistemas de Grid Computing [Berman et al., 2003], pasando por sistemas de Desktop Grid [Kacsuk et al., 2007]. Entre todos estos sistemas podemos destacar algunas soluciones como Globus en sistemas de Grid, Condor en sistemas de Clustering o Boinc en cuanto a sistemas de Desktop Grid. Todos estos sistemas, sin embargo, coinciden en un punto: todos están centralizados en algún punto. Esta centralización bien sea para facilitar tareas de administración y mantenimiento o bien sea por cuestiones corporativas le priva al sistema de algunas de sus más preciadas características en cuanto a soporte de fallos o escalabilidad ya que es precisamente este punto central el eslabón más débil de la cadena y por donde el sistema entero puede fallar.

Un sistema de computación mediante redes P2P supera esta desventaja al otorgarle a sus integrantes las labores administrativas y al darles iguales derechos ante el resto de los pares de la red. Sin embargo se deben solventar una serie de problemas técnicos sobre los que se está trabajando en la actualidad.

- **Volatilidad**: La gran volatilidad de un sistema P2P hace que sea necesario implementar algún sistema de checkpointing que permita parar la ejecución de un proceso en un par y retomarla en otro distinto. Existen diferentes soluciones y avances en sistemas de grid [Chtepen et al., 2009] y en sistemas de tipo Open-Mosix [Ho et al., 2008]. Entre dichas soluciones podemos encontrar varios sistemas de checkpointing utilizados actualmente: CHPOX[1] usado por OpenMosix, DMTCP[2] o libckpt[3]. En [Gil et al., 2009] se evalúa el impacto de aplicar Checkpointing a la computación distribuida

[1]The chprox web site. http://freshmeat.net/projects/chprox/
[2]The DMTCP web site. http://dmtc.sourceforge.net/
[3]The libckpt web site. http://www.cs.utk.edu/ plank/plank/www/libckpt.html

en redes P2P y sistemas de Grid computing. En dicho artículo se propone la realización de Checkpointings periódicos durante la ejecución de una tarea con objeto de utilizar las imágenes intermedias de la ejecución como puntos de partida para otros pares, replicando estas subtareas como medida de backup. Otra solución propuesta en [Zhang et al., 2009] consiste en predecir el tiempo que un par estará conectado a un sistema P2P y adaptar el scheduling de dichos sistemas a dichas predicciones maximizando así el número de ejecuciones finalizadas.

- **Heterogeneidad de sistemas**: En un sistema de computación voluntaria como sería un sistema de computación en redes P2P se debe tener en cuenta la heterogeneidad de sistemas que se conectarán a la misma, no sólo ya en cuanto a hardware si no también a nivel de sistema operativo o de librerías presentes en el sistema. Existen dos soluciones a priori a este problema: o bien se dividen los usuarios por grupos de pares o bien se utiliza virtualización. Existen propuestas para usar virtualización en sistemas de tipo Boinc [González et al., 2009] o de tipo Grid [Santhanam et al., 2005] en las cuales se propone que cada tarea a ejecutar en el sistema sea una instancia de una máquina virtual. Sin embargo, la pérdida de rendimiento en cuanto a carga de CPU o en cuanto a ancho de banda no parece justificar esta medida. En [Domingues et al., 2009] se ha estudiado el impacto que tiene una aplicación ejecutada sobre una máquina virtual y el impacto causado sobre sistemas de Grid Computing por usar virtualización. Tal vez, una solución sea el uso de virtualización sólo en el caso de que sea necesario, es decir, se podrían definir una serie de plataformas estándares y que cada par contara con un supervisor con dichas plataformas en hibernación de tal forma que si se da el caso de que no exista ningún par con el sistema operativo necesario para ejecutar una tarea que precise dicho sistema operativo, se elija un par libre del sistema y dicho par ejecute la tarea en la instancia estándar del sistema operativo requerido que tenía en hibernación. Con esto se ganaría en cuanto a ancho de banda, ya que no habría que transferir toda la máquina virtual a ser ejecutada si no sólo la aplicación. Además se ganaría en tiempo de arranque de la tarea ya que es mucho más rápido despertar una máquina virtual hibernada que arrancarla desde cero. Sobre este aspecto, el sistema de Grid Computing Entropia [Chien et al., 2003] propone el uso de una tecnología de máquinas virtuales propia para distribuir tareas en un sistema distribuido, utilizando para ello máquinas virtuales estándar y distribuyendo solamente los ejecutables de la tarea a ejecutar. Otro punto importante para solucionar el problema de la heterogeneidad de sistemas, es crear sistemas que permitan definir una tarea junto con todos sus parámetros. A este referente se ha propuesto el uso de GenWrapper [Marosi et al., 2009], un entorno de shell basado en POSIX permitiendo definir tareas a lanzar en sistemas de Grid Computing y Desktop Computing.
- Una de las cuestiones más importantes es cómo asegurar que lo que un par ha devuelto como resultado de la ejecución de una tarea es correcto. Ante este problema se pueden instaurar sistemas de créditos para poder puntuar el comportamiento de un par en el sistema, siendo de esta forma posible desterrar pares no deseados con el consenso de la mayoría. En [Zhao et al., 2005] se propone insertar en las tareas una serie de tareas *quiz* fáciles de verificar para comprobar el nivel de confianza que se puede poner en los resultados provenientes de un par determinado. En la literatura se menciona sistemas basados en la reputación como la solución a este problema. Diversos métodos se han propuesto para evaluar la reputación de los pares participantes en un sistema P2P [Xie and Bi, 2008][Heien et al., 2009].

III. Arquitectura de los Sistemas P2P

Los sistemas P2P [Domingues et al., 2009] también son redes superpuestas, es decir, se forman estableciendo conexiones lógicas entre los pares sobre la capa de Internet. Según la arquitectura de la red superpuesta, es decir, según la estructura del sistema a través del cuál se establecen las conexiones entre los pares, los sistemas P2P pueden clasificarse en estructurados y no-estructurados.

A. Sistemas P2P no-estructurados

En estos sistemas las interconexiones entre los pares se establecen de acuerdo a reglas flexibles, de manera jerárquica [Lua et al., 2005] o por medio de un servidor. A continuación se muestra cómo se subclasifican. En las Figuras 1, 2 y 3 se puede apreciar un ejemplo gráfico de la topología de estos sistemas:

- **Redes P2P centralizadas**: Fueron las primeras en aparecer, se hacen populares en sistemas como Napster [Yianilos and Sobti, 2001] alrededor del año 1998. Emplean un servidor que posee las referencias a los recursos que comparten los pares. Cabe resaltar que el intercambio de recursos se realiza de forma directa entre los pares. La topología típica de estos sistemas es una red estrella. En la Figura 1 se muestra un ejemplo de red P2P centralizada. Las líneas segmentadas representan los enlaces lógicos al servidor, las líneas continuas representan la comunicación entre pares y las flechas representan una consulta. Ésta consiste en que un par solicita un recurso al servidor y éste le responde con una dirección (paso 1) y después se comunica directamente con el par que posee dicho recurso (paso 2).
- **Redes P2P puras**: Estos sistemas no emplean ningún servicio centralizado. Cronológicamente aparecieron a continuación de las redes P2P centralizadas en sistemas como la primera versión de Gnutella [Yianilos and Sobti, 2001], del año 2000. Las conexiones entre los pares se establecen cuando un par se comunica con sus vecinos. Los pares vecinos se definen empleando algún criterio de proximidad.

La topología de estos sistemas es un grafo aleatorio. La Figura 2 ilustra un ejemplo de este tipo de red. Las líneas segmentadas representan los enlaces lógicos entre pares vecinos y las líneas continuas representan un proceso de comunicación. Las flechas indican algún proceso de intercambio de recursos.

- **Redes P2P híbridas**: Son una combinación de los enfoques anteriores en la que los pares se organizan de manera jerárquica clasificándolos en pares y superpares. Los pares se conectan a los superpares y los superpares se conectan a otros de su misma índole formando una red. Los superpares son entidades centrales dinámicas que actúan como servidores y que pueden ser sustituidas por otros pares si es que fallan o se desconectan. Generalmente, son elegidos por las características de su hardware. Los superpares tienen la función de gestionar las consultas de los pares que se han conectado a ellos y hacérselas llegar a otros superpares si es necesario. También, actúan como los pares comunes realizando consultas, compartiendo sus recursos o haciendo uso de los recursos de otros pares. Morpheus [Pourebrahimi et al., 2005] y la segunda versión de Gnutella [Pourebrahimi et al., 2005] son ejemplos de estos sistemas, aparecieron alrededor del año 2002. La Figura 3 representa una red híbrida. Se puede apreciar como los dispositivos con menor hardware (PDA y teléfono móvil) se conectan a los dispositivos con mayor hardware (computadora portátil). Las líneas continuas representan la conexión lógica entre par y superpar, las líneas segmentadas indican la conexión entre superpares y la línea punteada la comunicación entre pares. El paso uno implica la solicitud de un recurso de par a superpar. El paso dos representa la difusión de la consulta entre la red de superpares. En el paso tres, el par que inicio la consulta recibe la dirección del par que tiene el recurso que solicitó y comienza el intercambio de éste.

B. Sistemas P2P estructurados

En estos sistemas las conexiones entre los pares se establecen siguiendo el criterio de algún índice distribuido que es soportado por todos los pares en el sistema. Definimos índice distribuido de la siguiente manera:

Definición de índice distribuido P2P. Colección de apuntadores que son almacenados de manera simétrica por todos los pares del sistema. Permite organizar y administrar (buscar, dar de alta o dar de baja) los recursos que se comparten en la red P2P a través de indirección.

Normalmente, los índices son una TDD. En consecuencia, la ubicación de los pares y del contenido es pseudoaleatoria porque los pares emplean algún mecanismo determinista que los coloca en alguna posición fija dentro de un espacio de claves. Cada par almacena una pequeña tabla de encaminamiento que contiene al menos el identificador y la dirección IP de pares que cumplen con alguna medida de proximidad entre sus identificadores [Lua et al., 2005]. Estos sistemas fueron desarrollados por la comunidad científica y entre algunos de ellos se encuentran *Chord* [Stoica et al., 2001], *Pastry* [Rowstron and Druschel, 2001], CAN [Ratnasamy et al., 2001], *Kademlia* [Maymounkov and Mazieres, 2002], Tapestry [Zhao et al., 2001], Kelips [Gupta et al., 2003], Koorde [Kaashoek and Karger, 2003] y Viceroy [Malkhi et al., 2002]. La topología de estos sistemas puede ser representada como un red superpuesta anillo, árbol, mariposa y otras.

La Figura 4 ejemplifica una red estructurada tipo anillo. Los identificadores de los pares son las letras minúsculas. Las líneas grises representan los apuntadores hacia los dos siguientes pares según el orden del abecedario. Las flechas en negro ilustran un proceso de solicitud donde a pide un recurso de e (pasos uno y dos). Cuando a recibe la dirección de e se comunica directamente con él y comienza el intercambio de recursos descrito por el paso tres y representado por la flecha segmentada.

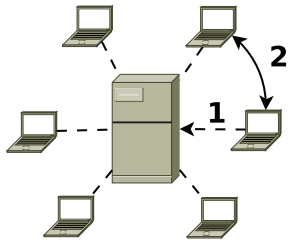

Fig. 1. Red P2P centralizada.

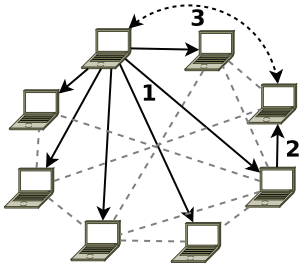

Fig. 2. Red P2P pura.

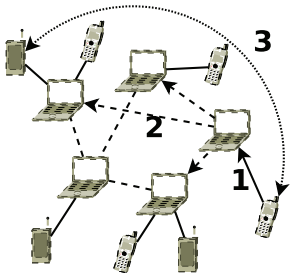

Fig. 3. Red P2P híbrida.

IV. Protocolos de los Sistemas P2P

En esta Sección vamos a tratar algunos de los protocolos de red P2P más extendidos [CHÁVEZ,]. Se entiende por protocolo de encaminamiento como el método para dirigir las

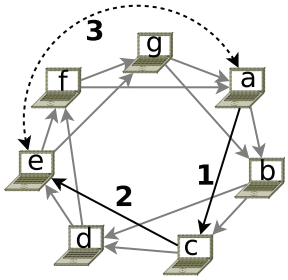

Fig. 4. Red P2P estructurada.

consultas realizadas por los pares, es decir, el funcionamiento del mecanismo de búsqueda.

A. *Chord (Anillo)*

Según sus autores, el objetivo principal de *Chord* es ofrecer una función de búsqueda eficiente para encontrar los recursos del sistema. El espacio de claves se organiza de manera circular y en orden ascendente, por lo que la topología de la red superpuesta es un anillo. La función de búsqueda se encarga de encontrar los recursos siguiendo un mecanismo de búsqueda unidireccional en el que las consultas se encaminan en sentido horario. Para hacer escalable y rápido al mecanismo de búsqueda, se emplean enlaces lógicos que incrementan su longitud en potencias de dos. Para mantener coherente al índice, *Chord* hace uso de un protocolo de estabilización para autoorganizar los enlaces correspondientes a los pares sucesores y un protocolo de reparación que mantiene actualizados los enlaces de potencias de dos.

Tanto los recursos como los pares comparten el mismo espacio de claves y éstas se obtienen aplicando una función de dispersión criptográfica, como Secure Hash Standard (SHA-1 [Eastlake and Jones, 2001]), al recurso a compartir o a la dirección IP del par. La longitud de los identificadores es de m bits y éstos son arreglados en un espacio circular módulo 2^m, formando una red superpuesta tipo anillo. Las claves de los recursos se almacenan por el par cuyo identificador es mayor o igual que la clave de los recursos en cuestión; a este par se le conoce como sucesor. Por ejemplo, en la Figura 5 se muestra un espacio de claves *Chord* de tamaño 2^6. Con esta configuración se obtienen identificadores de longitud $m = 6$ bits que están en el rango $[0, 63]$ en notación decimal. Los identificadores de los recursos están representados con cuadros, mientras que los identificadores de los pares están representados por círculos. En este ejemplo, las fechas que salen de los identificadores de los recursos apuntan al par sucesor que las almacena. Se puede apreciar que las claves 62, 0 y 3 se almacenan por el par con identificador 7, debido a que la aritmética es módulo 2^6.

En *Chord*, los pares guardan una pequeña cantidad de referencias a otros pares en una *tabla de apuntadores* (finger table) y en una *lista de sucesores* (successor list). El almacenamiento en la tabla de apuntadores es del orden $O(logN)$, donde N es el número de pares. Por ejemplo, para un espacio de 2^m identificadores cada par almacena m referencias que están compuestas por el identificador del par, su dirección IP y

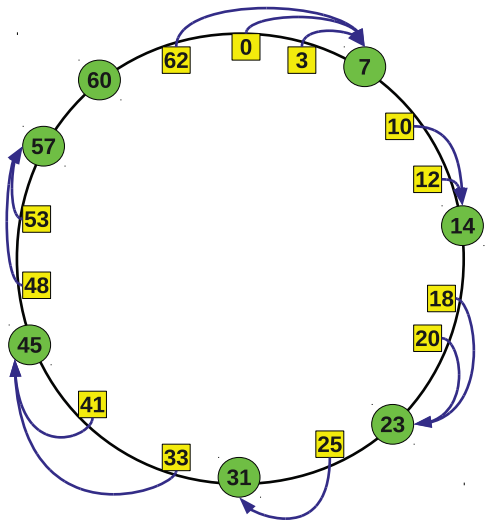

Fig. 5. Espacio de claves *Chord* tamaño 2^6.

puerto UDP. Para el par n tenemos que la fila i de su tabla contiene el identificador del par sucesor correspondiente a la clave $n + 2^{i-1}$, donde n es el identificador del par en cuestión e $i \in [1, m]$. Por otro lado, la lista de sucesores contiene normalmente m referencias que apuntan a los m pares sucesores del par n. También se mantiene un apuntador al par predecesor inmediato para facilitar la autoorganización del sistema. En conjunto, el estado de un par *Chord* tiene una complejidad de $O(2 * logN)$, donde N es el número de pares. La Figura 6 representa el estado del par 7; éste posee una tabla de apuntadores de 6 entradas, debido a que el tamaño del espacio de claves es de 2^6, y una lista de sucesores de tamaño seis. Las líneas segmentadas apuntan a los pares de los que tiene conocimiento el par 7 a través de su tabla. Podemos observar que la proximidad entre el identificador del par siete y los identificadores de los pares almacenados en su tabla de apuntadores aumenta en potencias de dos.

Debido a la organización del espacio de claves en forma de anillo, cada par *Chord* es consciente de al menos la dirección de su sucesor inmediato. Las consultas por los recursos son reexpedidas, en orden ascendente con respecto a las claves, por los pares sucesores hasta alcanzar al par que posee la referencia del recurso deseado. Bajo este esquema, se tiene un mecanismo de búsqueda ineficiente con complejidad $O(N)$, donde N es el número de pares participantes.

Para hacer escalable el mecanismo de búsqueda se emplea la información de la tabla de apuntadores. Como se explicó en el apartado correspondiente al estado del par, cada entrada en la tabla de apuntadores almacena la referencia a pares cuya proximidad entre identificadores se incrementa en potencias de dos, por lo que los pares poseen enlaces de gran distancia lógica a otros pares. Cuando un par realiza o recibe una consulta por la clave k de un recurso, éste emplea su tabla de apuntadores y reenvía k al par cuyo identificador sea menor que k y cuya proximidad sea máxima. La consulta se reexpide siguiendo el criterio anterior hasta que se alcanza al par que conoce al par sucesor del recurso k, es decir, el

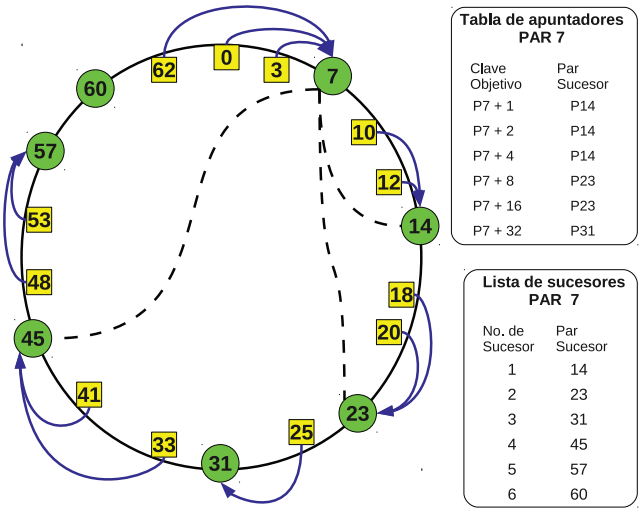

Fig. 6. Estado del par siete en un espacio de claves *Chord* tamaño 2^6.

que almacena la referencia al recurso. Por ejemplo, la Figura 7 ilustra el mecanismo de búsqueda mejorado empleando la tabla de apuntadores. El par 7 realiza una consulta por el par 60, así que envía la consulta al par 45 ya que 45 es menor que 60. Siguiendo el mismo criterio, 45 reenvía la consulta al par 57, ya que 57 es menor que 60 y de entre todas las entradas de la tabla de apuntadores de 45 es la que representa mayor proximidad. En ese punto finaliza la búsqueda ya que 57 sabe que 60 es su sucesor. Las líneas dobles entre los pares 7, 45 y 57 representan el camino tomado por la consulta generada por el par 7. Figura 7: Búsqueda empleando lista de apuntadores, el par 7 realiza una consulta para encontrar el par 60.

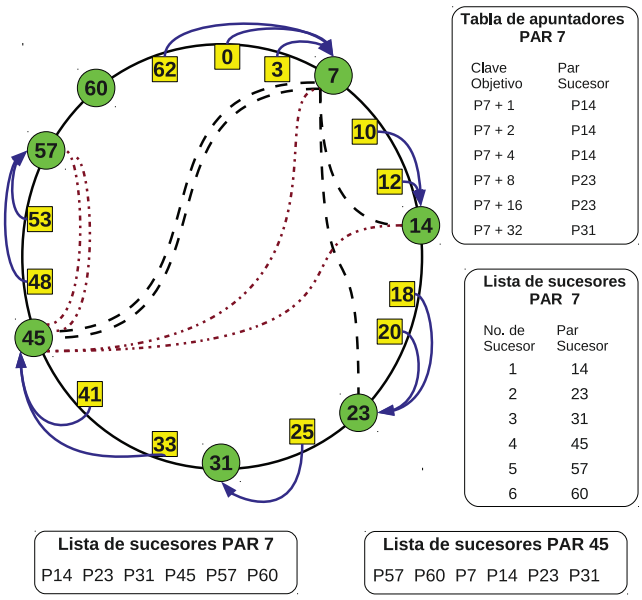

Fig. 7. Búsqueda empleando lista de apuntadores, el par 7 realiza una consulta para encontarr el par 60.

Para que las entradas y salidas de los pares tengan el mínimo efecto en la tabla de dispersión distribuida se hace uso de dispersión consistente. La dispersión consistente [Karger et al., 1997] (consistent hashing) es aquello que permite que añadir o quitar alguna ranura de una tabla de dispersión implique que, en promedio, solo $K = r$ claves sean reasignadas, donde K es el número total de claves y r el número de ranuras, en este caso número de pares. En una tabla de dispersión común, un cambio en su tamaño implica que todas las claves sean reasignadas. Para que un par n se una al índice *Chord*, primero debe obtener un identificador, por simplicidad puede elegir uno de manera aleatoria, procurando que el espacio de claves sea lo suficientemente grande para evitar colisiones o, mediante el uso de la función SHA-1 u otra función de dispersión criptográfica. En seguida, mediante algún mecanismo externo, el par n realiza una búsqueda preguntando por su propio identificador. El resultado de la búsqueda es la referencia a su par sucesor s. Posteriormente, s actualiza el valor de su apuntador predecesor con n. Después, n comienza a llenar su tabla de apuntadores preguntando a s la identidad de los pares sucesores de las claves $n + 2^{i-1}$, correspondientes a las entradas de la tabla de apuntadores. Hasta este punto, n posee el apuntador a su sucesor inmediato y la tabla de apuntadores, pero los demás pares, a excepción de s, no conocen la presencia de n.

Chord emplea un protocolo de estabilización para verificar si el apuntador al sucesor inmediato es correcto. Éste consiste en que un par k pregunta a su sucesor inmediato s la identidad de su par predecesor inmediato p. Si k y p son iguales quiere decir que el apuntador al sucesor inmediato es correcto. En caso contrario, es decir p diferente de k, quiere decir que un nuevo par ha llegado al índice *Chord*, así que k actualiza su apuntador al sucesor inmediato con la referencia a p y comunica a p que k es su predecesor. El protocolo de estabilización también se emplea para mantener actualizada la lista de sucesores y se ejecuta de manera periódica, pero en caso de ser necesario puede invocarse.

Para que los otros pares estén conscientes de la llegada del par n, el predecesor de n debe ejecutar el protocolo de estabilización para que n le dé valor a su apuntador correspondiente al predecesor inmediato p. En ese momento n copia las claves de los recursos que están entre n y p. Por último, el sucesor inmediato s libera las claves de n.

En este punto, el par entrante n puede encaminar consultas procedentes de su predecesor. Solo s y p conocen a n, por lo que las tablas de apuntadores de los otros pares están inconsistentes. Las entradas de las tablas de apuntadores se actualizan mediante un protocolo de reparación que pregunta, a menor cadencia que el protocolo de estabilización, la identidad del par correspondiente a la clave $ID + 2^{i-1}$. El protocolo puede ser invocado en caso de ser necesario.

Las salidas informadas se manejan de manera similar a la llegada de un par. Simplemente el par n que abandona el índice le comunica a su sucesor s su salida, entonces n transfiere las claves de los recursos que poseía a s y le hace saber la identidad de su nuevo predecesor inmediato. Los autores de *Chord* mencionan que para asegurar que el índice funcione correctamente es necesario que tanto la referencia al par sucesor como las claves de los recursos que almacena

cada par sean coherentes. Y para obtener búsquedas rápidas se requiere que la tabla de apuntadores se mantenga actualizada.

En *Chord*, se hace uso de temporizadores para determinar si un par ha fallado o se retiró del sistema sin informar su salida. Si el par n falla y n está presente en la tabla de apuntadores de otros pares, entonces, los pares que contienen a n deberán buscar al sucesor de n. Sin embargo, la transitoriedad de los pares puede ocasionar que las referencias a los pares contenidos en la tabla de apuntadores estén desactualizadas. En caso de que el sucesor inmediato de n falle, entonces n reemplazará el apuntador por el segundo par en su lista y así sucesivamente. Con esto se obtendrán búsquedas lentas pero correctas y después de cierto tiempo, la tabla de apuntadores se actualizará por medio del protocolo de reparación. Bajo este contexto, el anillo *Chord* falla solo si todos los pares en la lista de sucesores fallan.

B. Kademlia

Kademlia [Maymounkov and Mazieres, 2002] fue propuesto en el 2002. Su espacio de claves es organizado como un árbol binario lleno donde se asignan las claves de los recursos a los pares más próximos empleando la función lógica XOR como medida. El mecanismo de búsqueda sigue un enfoque asíncrono paralelo y además cada vez que se recibe un mensaje se actualiza el estado del par. Ésto hace que el estado de los pares de *Kademlia* se mantenga actualizados en la medida del intercambio de mensajes.

Las referencias a pares y recursos se organizan empleando un árbol binario lleno y su posición en éste se determina por el prefijo más corto. Para ubicar su posición, los pares recorren el árbol en profundidad siguiendo la ruta donde su identificador no está presente.

Las claves de los recursos y pares se obtienen de manera aleatoria, si es que el espacio de claves es lo suficientemente grande para reducir la probabilidad de colisiones, o aplicando una función de dispersión criptográfica. Las claves de los recursos son almacenadas por el par más próximo. La proximidad entre claves está definida por el resultado de aplicar la función lógica XOR a las claves en cuestión. Por ejemplo, la clave k_1 = 1000 es almacenada por el par p_1 = 1011 en lugar del par p_2 = 0011 ya que p_1 XOR k_1 tiene un resultado menor que el que se obtiene con p_2. La longitud típica de las claves en *Kademlia* es de 160 bits. La Figura 8 ilustra un espacio de 24 claves e identificadores de 4 bits. Se puede apreciar cómo el par p con identificador 1010 se queda en el tercer nivel del árbol debido a la longitud de su prefijo. Si llegara un par q con identificador 1011, entonces p y q descenderían al cuarto nivel para seguir teniendo un árbol lleno.

Los pares *Kademlia* almacenan referencias a otros pares empleando una especie de lista, que los autores llaman $k-buckets$ y son de tamaño k. De manera general, las claves tienen una longitud de l bits, así que por cada bit en el identificador se tiene un $k-bucket$. Por ejemplo, en la Figura 9 se tiene un espacio de claves de tamaño 16, representadas por cuatro bits. Es por ello que para el par con clave 1100 se tienen cuatro $K-bucket$.

Cada $k-bucket$ almacena identificadores con una proximidad específica. Sea i el i-ésimo $k-bucket$ correspondiente al

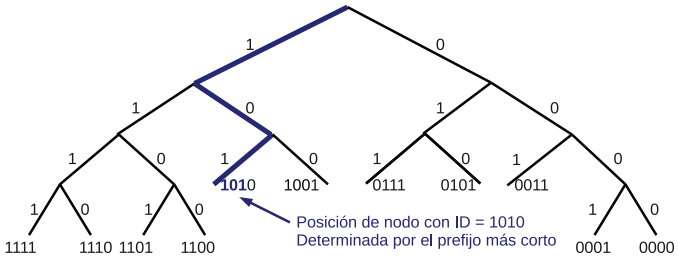

Fig. 8. Espacio *Kademlia* con claves de 4 bits.

bit i de un clave con longitud l. Entonces, tendremos que el i-ésimo $k-bucket$ almacena referencias con proximidad entre 2^i y $2^{i+1}-1$. En la Figura 9, se tiene que la proximidad entre la clave 1100 y las claves contenidas entre en el $bucket_0$ está entre 2^0 y 2^0-1. Para el $bucket_3$, la proximidad entre claves está entre 8 y 15.

También hay que notar que cada $k-bucket$ representa un subárbol y que, para cualquier subárbol dado i y pares x y y contenidos en el mismo subárbol, la proximidad entre x y y siempre es mayor que la proximidad entre x y un par z, donde z está contenido en un subárbol distinto. Lo anterior se aprecia en la Figura 9, ya que las claves contenidas en el $bucket_1$ son más próximas entre si, en comparación a la clave almacenada en el $bucket_0$.

Otro punto a considerar es que a medida que la proximidad entre los pares disminuye, es más difícil llenar los $k-buckets$. *Kademlia* se asegura que cada par conozca al menos un par de cada subárbol, es decir, que al menos exista una referencia a un par en cada $k-bucket$.

En *Kademlia* las búsquedas se realizan siguiendo un mecanismo asíncrono paralelo y para ello se emplean las referencias contenidas en los $k-buckets$. Cuando un par P quiere realizar una consulta por un recurso con clave c, P obtiene α pares próximos con respecto a la medida XOR y les envía la consulta por c. Cuando los α pares reciben la consulta por c, si poseen la referencia a c se la hacen llegar a P, en caso contrario consultarán sus $k-buckets$ y regresarán a P la referencia al par más cercano a c que conocen. La consulta se reexpide de manera iterativa y en cada iteración, la proximidad entre las claves se reduce al menos en un medio.

En la Figura 10, el par con identificador 1100 genera una consulta por la clave 0011. Siguiendo el protocolo, 1100 emplea la información de sus $k-buckets$ y encuentra que con el par 0001, localizado en el $k-bucket$ correspondiente al bit 3, se genera la menor proximidad, así que le envía la consulta. El par 0001, empleando sus $k-buckets$ responde con la clave 0011 y la búsqueda finaliza.

En *Kademlia* se sigue un enfoque dinámico para la detección de fallos y el mantenimiento del estado, ya que estas acciones dependen del mecanismo de búsqueda y del tráfico de la red. Ésto se debe a que las consultas de *Kademlia* emplean $piggybacking(acarreoacuestas)$; al realizar una consulta se adjunta a ésta la información necesaria para actualizar los $k-buckets$ de los pares que la reciban.

Los $k-buckets$ se ordenan según la aparición de los pares, es decir, del último par visto hasta el par visto más

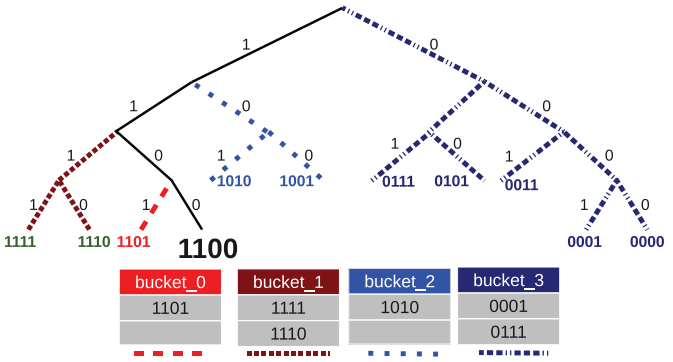

Fig. 9. Estado del par *Kademlia* 1100.

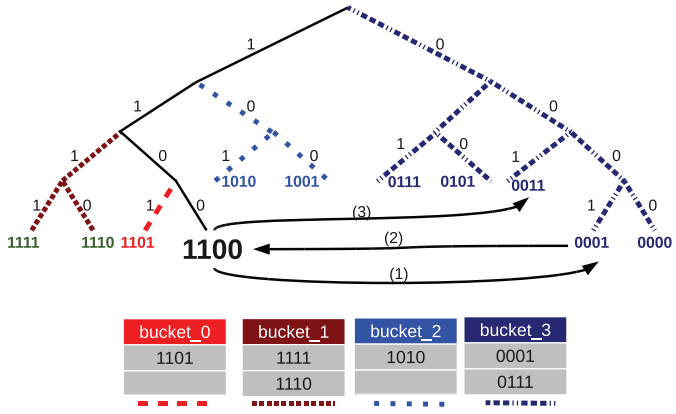

Fig. 10. Búsqueda con α = 1 por la clave 0011, generada por 1100.

recientemente, colocando el último par en la cabeza de la lista y el más reciente en la cola. Cada vez que un par P recibe un mensaje de otro par Q, P actualiza el $k-bucket$ correspondiente a Q y se realiza alguna de las siguientes acciones:

1) Si Q ya estaba en el $k-bucket$ de P entonces Q se mueve a la cola, en otro caso:
2) Si aún queda espacio en el $k-bucket$ simplemente se agrega la referencia a Q, colocando a éste en la cola, en otro caso:
   - P verifica la vivacidad del último par visto R, si éste responde entonces R se mueve a la cola del $k-bucket$ y se descarta el registro de Q, en caso contrario,
   - Se elimina R del $k-bucket$ y se coloca a Q en la cola.

Como se puede apreciar, los pares vivos nunca son eliminados de los $k-bucket$, lo que hace que *Kademlia* resista ciertos ataques de denegación de servicio y que considere en su diseño a los pares con tiempos de sesión mayor.

Cuando el tráfico disminuye, puede ocurrir que el estado de los pares se torne inconsistente, por lo que los pares *Kademlia* actualizan sus $k-buckets$ cada hora mediante un protocolo de estabilización. Este protocolo implica elegir aleatoriamente alguna entrada en los $k-buckets$ para comprobar su permanencia, realizando una búsqueda por el identificador del par.

El proceso de unión se realiza de la siguiente forma. En principio, un par P debe conocer la identidad de otro par Q, entonces P inserta a Q en el $k-bucket$ apropiado. Después, realiza una búsqueda por su propio identificador. Posteriormente P actualiza los $k-buckets$ que están más allá de su vecindario. Por medio de la primera búsqueda y la actualización de los $k-buckets$, P llena su estado y se hace presente en los estados de otros pares, si es que lo requieren.

Las fallos se detectan cuando no se obtiene respuesta de alguno de los pares contenidos en los $k-buckets$. Cuando esto ocurre se remueve la entrada y se inserta en otra. Siempre que el tráfico de mensajes sea considerable, los $k-buckets$ se mantendrán actualizados.

C. Pastry

Pastry [Rowstron and Druschel, 2001] fue propuesto en el 2001. Se define como un sistema P2P completamente descentralizado, escalable y autoorganizado que puede usarse como sustrato para localizar objetos en una red superpuesta. Al igual que *Chord*, el espacio de claves está ordenado de manera circular, pero las claves de los recursos se asignan a los pares más cercanos. La función de búsqueda se implementa mediante encaminamiento Plaxton [Plaxton et al., 1999]. La autoorganización del sistema se ejecuta de manera perezosa al descubrir fallos en la resolución de las búsquedas.

Pastry usa un espacio circular de claves cuyo tamaño es de 2^l. Los recursos y pares comparten el mismo espacio de claves y éstas tienen una longitud de l bits, la cual se expresa en base 2^b. Los identificadores de los pares se eligen mediante una función de dispersión aplicada a la dirección IP del par o a una llave pública, de tal forma que las claves de los recursos y pares queden uniformemente distribuidas en el espacio de claves. Las claves de los recursos serán almacenadas por los pares cuyo identificador sea numéricamente más cercano a éstas. Una clave k_i es más cercana a otra clave k_j mientras mayor es el número de dígitos que comparten sus prefijos. Por ejemplo, con b = 4 y l = 28, la clave k_1 = ABC3491 es más próxima a la clave k_2 = ABC3476 que a la clave k_3 = AB54420, ya que el prefijo ABC34 que comparte k_1 y k_2 tiene más dígitos en común que el prefijo AB que tienen en común k_1 y k_3.

Cada par *Pastry* almacena referencias a otros pares en un conjunto de hojas, un vecindario y una tabla de encaminamiento, cada una con distintas características que se expresan a continuación.

i	Tabla de encaminamiento D			
0	**0**2212102		**2**2301203	**3**1203203
1		**11**301233	1**2**230203	1**3**021022
2	10**0**31203	10**1**32102		
3	102**0**0230	102**1**1302	102**2**3211	
4	10230322	10231000	10232121	
5	1023300**1**		1023323**2**	
6			1023312**0**	
7				

Conjunto de hojas H			
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Vecindario V			
00123223	32001212	01211232	11323311
01221232	02312123	22331111	11232321

Fig. 11. Estado del par 10233102 para un índice *Pastry*.

Sea D la tabla de encaminamiento de un par *Pastry*, como ejemplo mostramos la del par 10233102 representada en la

Figura 11. La tabla D está compuesta por i filas, que están en el rango $i \in [0, log_2 b(N) - 1]$, y j columnas que están en el rango $j \in [0, 2^b - 1]$. Tanto para i como para j, b es la base numérica de los identificadores, que para el caso de este ejemplo $b = 4$.

En la fila i se almacenan referencias a pares cuyo identificador posee un prefijo de longitud i. Además, para la fila i se tiene que el dígito $i + 1$ del identificador es igual al valor de la columna j, el resto de los dígitos es diferente. Si no existe un par que cumpla con el prefijo adecuado entonces la entrada se queda vacía. En la Figura 11, se resalta el dígito correspondiente a la columna j.

La proximidad de los identificadores se incrementa en proporción a i, por ejemplo, usando la Figura 11, podemos notar que el identificador 10233232, que está en la fila $i = 5$, es más próximo al identificador del par 10233102, que 22301203, contenido en la fila $i = 0$. Cada entrada en la tabla de encaminamiento tiene, además del identificador del par, su dirección IP.

Se puede apreciar que la tabla de encaminamiento D sigue el formato $prefijo - columna - restodelidentificador$. Por ejemplo, para la entrada contenida en la columna 1 fila 4 se tiene 1023-1-000, así que 1023 es el prefijo común, 1 el número de columna y los demás dígitos son el resto del identificador.

El conjunto de hojas H mantiene pares cuyo identificador es numéricamente cercano y su cardinal se representa con $|H|$. Su tamaño recomendado es de 2^b o 2^{b+1}. En H, la primera mitad de las referencias en el conjunto de hojas $H_{-|H|/2}$ son numéricamente menores al identificador del par; la otra mitad $H_{+|H|/2}$ son numéricamente mayores.

El vecindario V almacena referencias a pares que son cercanos espacialmente y regularmente no se usa para encaminar mensajes. Normalmente es de tamaño 2^b o 2^{b+1} y su cardinal se representa con $|V|$.

Para encaminar una clave k, un par *Pastry* p hace uso de las referencias contenidas en su tabla de encaminamiento D y conjunto de hojas H. En principio, cuando p recibe una consulta por k, verifica si k está en el rango de su conjunto de hojas; si es así, la consulta se envía al par con identificador numéricamente más cercano. En este caso la búsqueda finalizaría ya que el par contenido en H debe poseer la referencia al recurso con clave k.

Si k se encuentra fuera del rango de H entonces se emplea la tabla de encaminamiento D. En este caso el par p obtiene la longitud del prefijo que tiene en común con k mediante $l = shl(p, k)$. La función $shl(p, k)$ devuelve la longitud del prefijo, en dígitos, compartida entre la clave p y k. Después, p envía la consulta al par q contenido en una fila mayor o igual que l y cuya longitud de prefijo en común $l' = shl(q, k)$ sea mayor que l, es decir, $l' > l$.

La Figura 12 ilustra el proceso de encaminamiento en un índice *Pastry*. El par 10233102, haciendo uso de su tabla de encaminamiento (ver Figura 11), genera una consulta por la clave 32211331 y elige al par $D_{0,3}$ = 31203203 para dirigir la consulta. Los cuadros representan a las claves de recursos y los círculos a los pares. Las líneas que unen cuadros y círculos indican qué par o pares poseen la referencia a un recurso dado. Las líneas punteadas describen la ruta que tomó la consulta. Los números en negritas representan el prefijo común entre par y clave. Se puede observar como la consulta se reexpide a pares que poseen un prefijo en común con la clave y como, en cada salto, la longitud del prefijo crece.

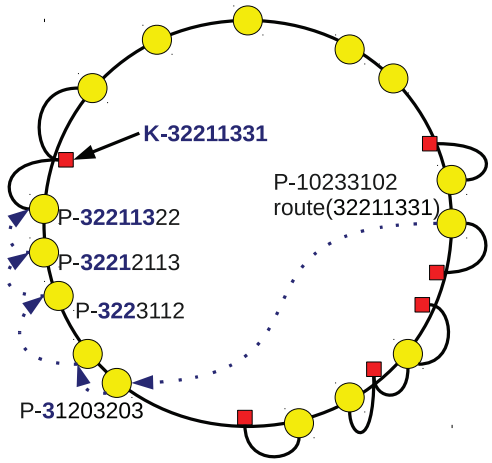

Fig. 12. Direccionamiento Plaxton en *Pastry*. Consulta generada por el par 10233102 para hallar la clave 32211331.

En *Pastry*, el procedimiento de unión al índice se realiza de la siguiente manera. Primero, un nuevo par obtiene su identificador mediante el uso de una función de dispersión criptográfica, por ejemplo, aplicando la función SHA-1 a la dirección IP o a una clave pública.

Después, haciendo uso de algún mecanismo de autoarranque, un par con identificador P, se comunica con un par con identificador K que se encuentra ya integrado al sistema. Para inicializar su estado, P envía un mensaje de unión a K y K reenvía el mensaje que pasa a través de los pares n_1, n_2,... hasta n_i, donde n_i es el par numéricamente más cercano a P. En el proceso de unión, P recibirá, como respuesta de su mensaje de unión, información para llenar su estado de la siguiente manera:

- P recibe el vecindario del par K ya que K es próximo a P de acuerdo a una medida de proximidad de red y, bajo el caso de que K y P no compartan un prefijo, también recibe la fila cero de la tabla de encaminamiento D.
- P recibe la fila uno de la tabla de encaminamiento de n_1, ya que, debido al protocolo de encaminamiento, P y n_1 deben tener un dígito en común en su prefijo. Del par n_2, P recibe la fila dos y así sucesivamente hasta completar la fila i por medio del par n_i.
- Por último, ya que n_i y P son numéricamente próximos, ni envía su conjunto de hojas a P y P informa su presencia a los pares contenidos en su estado. En este punto el par P se encuentra integrado en el índice *Pastry*.

La información de encaminamiento enviada a nuevos pares lleva una etiqueta de tiempo. Al momento de completar su estado, el par nuevo regresa su estado a los pares que participaron en el proceso de unión. Éstos verificarán que las etiquetas de tiempo concuerden y en caso de que éstas no

empaten, se le solicita al nuevo par que vuelva a comenzar el proceso de unión.

Las salidas informadas y no informadas se toman por igual y se detectan perezosamente cuando se encamina una consulta y no se obtiene una respuesta.

Para reemplazar un par contenido en el conjunto de hojas H se envía una consulta al par con mayor índice solicitando su conjunto de hojas H'. Las referencias almacenadas en H' también son válidas para H ya que son cercanas numéricamente y sustituye la entrada fallida por la mejor referencia contenida en H'. Posteriormente se verifica la vivacidad de la referencia seleccionada.

Para reemplazar las entradas ij de la tabla de encaminamiento D de un par P, se solicita a algún par N contenido en la fila i su fila i'. Debido a que i' es una fila válida para D, el par puede copiar la fila entera y verificar la vivacidad de cada referencia. En caso de que la referencia no funcione, se solicita a un par diferente M su fila i''. Con alta probabilidad, este mecanismo asegura que se encontrará un reemplazo adecuado para la entrada ij fallida de D, si es que existe.

Las fallas en el vecindario V se reparan periódicamente verificando la vivacidad de las entradas y con el mismo mecanismo de reparación de H. No se sigue el enfoque perezoso ya que V no se emplea con fines de encaminamiento.

D. CAN

Content Addressable Network (CAN) propuesta por primera vez en [Ratnasamy et al., 2001] es un sistema P2P que provee funcionalidades de tabla de dispersión de forma distribuida. CAN fué diseñado para ser escalable, tolerante a fallos y autorganizada. El diseño básico de su arquitectura es un espacio de coordenadas Cartesianas multidimensional sobre un toro, siendo *d-dimensional* el espacio lógico sobre el que se mapean las claves del sistema y los pares que lo componen. Según [Gummadi et al., 2003], si el número *d* de dimensiones es logN, donde N es el número de pares del sistema, podemos considerar que CAN sigue una topología de hipercubo logN-dimensional. En este espacio de coordenadas a cada par se le asigna una partición del espacio, de tal forma que a cada par le corresponde una zona única y diferenciada. En CAN, un par mantiene una tabla de enrutado con las direcciones IP y la zona de coordenadas virtual que le corresponden a sus vecinos en el espacio de coordenadas. Usando estas coordenadas, un par es capaz de enrutar un mensaje hacia su destino usando un simple algoritmo voraz que reenvía dicho mensaje hacia aquel de entre sus vecinos que esta más cerca del destino en el sistema de coordenadas.

Como podemos ver en la Figura 13 extraída de [Ratnasamy et al., 2001], el espacio de coordenadas se usa para almacenar pares del tipo (clave, valor), para ello se usa una función determinista que a cada clave K posible se le asigna un punto P en el espacio de coordenadas usando una función de dispersión. El protocolo de búsqueda garantiza que cualquier par puede aplicar la misma función de dispersión y obtener el mismo punto P del espacio, de tal forma que ejecutando el protocolo de enrutado ya descrito es posible llegar al par responsable de dicho sector del espacio con un coste de $O(d * N^{1/d})$.

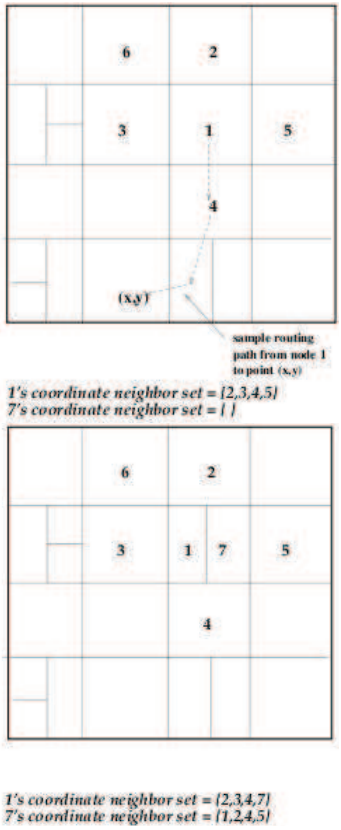

Fig. 13. Sistema CAN de 2D antes y después de que cierto par 7 entre al sistema. También se muestra el camino óptimo entre 1 y el punto (x, y).

Cuando un nuevo par *a* desea acceder al sistema, debe ejecutar el algoritmo de arranque, o *bootstrapping*, correspondiente. En CAN existe una mecanismo de DNS que permite localizar pares del sistema mediante un nombre DNS. Una vez que el par ha recuperado la IP de algún par del sistema, se pone en contacto con él indicándole su deseo de entrar a formar parte del sistema. Para ello, elige un punto al azar P del espacio de coordenadas y lo transmite al par del sistema, éste usa el protocolo de enrutado para enviar dicho mensaje al punto P de la red. Una vez llegue a su destino, el par q encargado de dicho punto dividirá su espacio de coordenadas en 2 quedandose él con una mitad y dándole la otra al nuevo par. Una vez a esté conectado a q, éste último comunica a a su tabla de vecinos con objeto de que a construya la lista de par vecinos de las regiones adyacentes a la suya.

En el caso de que algún par x abandone el sistema, existe un algoritmo de reestructuración encargado de que alguno de los vecinos del par x tome el control sobre la zona anteriormente controlada por x. Una vez hecho esto, el nuevo par encargado de la zona informa a sus vecinos del cambio para que estos mantengan actualizada su tabla de vecinos. El número de vecinos que posee un cierto par depende sólo del numero d de dimensiones que posea el sistema, por ejemplo $2 * d$, y nunca del número total de pares contenidos en el mismo.

Una forma sencilla de mejorar este sistema y de añadirle

replicación sería la de mantener un sistema con r sistemas de coordenadas, cada uno de los cuales se llama $reality$ o $realidad$. Cada par de este $multiverso$ poseería unas coordenadas distintas en cada una de las realidades con una lista de vecinos totalmente distinta para cada una de las realidades. De esta forma, cada par (clave, valor) que se insertase en el sistema correspondería a un área del espacio distinto para cada una de las realidades, siendo almacenado por r pares distintos.

V. Información espacial

Como ya se ha mencionado con anterioridad, el objetivo es tratar de gestionar información espacial en sistemas P2P. Por eso, en esta Sección vamos a hablar sobre información espacial con el propósito de adentrar al lector más en este campo, para ello nos hemos apoyado en [Zurita, 2011] donde se habla con detalle sobre ella. La *información geográfica* es aquella que permite modelar y representar los *fenómenos* espaciales o *entidades* del mundo real que configuran el amplio y variado espacio en el que se desarrolla la actividad humana: el territorio. Las entidades pueden ser *naturales*, como una montaña o un río, o *artificiales*, como un edificio o una carretera. Y no siempre son elementos tangibles en el espacio; pueden ser invisibles, como los límites administrativos.

Cualquiera que sea su tipo, las entidades poseen una serie de características comunes, que deben considerarse para la elaboración de cualquier modelo: tienen un tamaño y una forma determinada y se encuentran localizadas en un lugar concreto en el espacio. No es difícil imaginar que los recursos de la información alfanumérica tradicional resultan insuficientes para reflejar estas características espaciales en un modelo como el que se observa en la Figura 14.

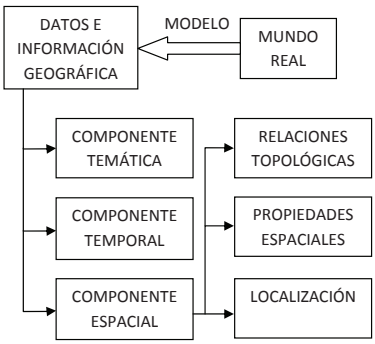

Fig. 14. Componentes de los datos e información geográfica.

Es necesario definir una representación de las entidades del *mundo real* que recoja al menos tres aspectos: la *localización*, respecto a un sistema de referencia bien definido; las propiedades espaciales, como el *tamaño* y la *forma*; y las *relaciones topológicas* entre objetos, como la proximidad, contigüidad o inclusión (este desglose puede ser observado con más claridad en la Figura 14). Estos tres aspectos conforman la denominada *componente espacial*, que es característica de los datos espaciales. En síntesis, la información geográfica es aquella que posee una *componente espacial*, además de la *componente temática* y *la componente temporal*.

La *componente* temática recoge aquellos aspectos descriptivos que se quieren vincular al objeto geográfico que representa a la entidad del mundo real; por ejemplo, el nombre de un río o el tipo de cultivo existente en una parcela. Los elementos temáticos pueden representar dos tipos de caracteres: cualitativos o *atributos* y cuantitativos o *variables*. Los atributos son elementos que no pueden medirse numéricamente como el tipo de cultivo existente en una parcela o si un municipio es cabeza de partido judicial. El hecho de que puedan ser codificados mediante valores numéricos para facilitar su tratamiento informático no altera su carácter cualitativo. Las variables, por su parte, almacenan valores medibles o, al menos, numerables. Cuando una variable sólo puede adoptar valores aislados, normalmente números enteros, se dice que es *discreta*; por ejemplo, el número de industrias existentes en un municipio. Pero si entre dos posibles valores de la variable pueden existir infinitos valores intermedios, se denomina *continua*; por ejemplo, la superficie de una parcela. Para que estas variables sean manejables es necesario reducir los infinitos valores que pueden adoptar, limitando la precisión utilizada. Es lo que se denomina *discretizar* una variable continua; por ejemplo, midiendo las superficies de las parcelas en metros cuadrados sin decimales. Las variables también se pueden clasificar en función del tiempo (que a su vez es una variable continua). Cuando se refieren a un instante de tiempo concreto se las denominan variables de *stock*; por ejemplo, un censo de población obtenido al 01/01/2011. Mientras que se registran los sucesos producidos durante un intervalo de tiempo, se llaman variables de *flujo*; por ejemplo, las altas a la Seguridad Social durante un año. Dos flujos de la misma variable pero en sentido contrario dan lugar a un *saldo*, como las altas y bajas a la Seguridad Social producidas durante un año. Esta clasificación resulta especialmente relevante porque pone en relación las componentes temática y temporal de la información espacial.

La *componente temporal* registra el instante o intervalo de tiempo en el que fueron capturados los datos, su período de validez o la periodicidad de su actualización. En realidad, las tres componentes de la información geográfica (espacial, temática y temporal) se encuentran relacionadas. Normalmente, resulta necesario asignar un valor (o un intervalo de variación prefijado) a dos de ellas para medir la tercera. Un ejemplo puede ser la elaboración de un *mapa de cultivos*, cuyo objetivo es registrar el tipo de cultivo (trigo, cebada, girasol, ...), existente en cada parcela de una determinada región. Para capturar la información del tipo de cultivo (componente temática) es necesario determinar previamente sobre qué parcelas se va a realizar el estudio (componente espacial) y en qué momento del año y cuánto va a durar dicho estudio (componente). Llegados a este punto, resulta pertinente aclarar una cuestión terminológica derivada de las componentes no espaciales de la información geográfica. Desde el momento que es posible asignar a los objetos espaciales atributos y variables temáticas de todo tipo (no siempre de carácter estrictamente *geográfico*), quizás sería más correcto hablar de *información territorial* y otro término similar en lugar de *geográfica*, puesto que, siendo realistas, se superan los límites de ésta. En cualquier caso, el término *información geográfica*

se encuentra tan arraigado que continuaremos empleándolo en lo sucesivo. En ocasiones, se alude a la *calidad* como la cuarta componente del dato y la información geográfica. Este punto de vista es válido desde el momento que la calidad es una característica intrínseca de los datos (porque éstos siempre contienen errores), aunque lo cierto es que la cantidad debe ser descrita para que resulte útil a la descripción se que realiza mediante *metadatos geoespaciales*, que estarían asociados al dato pero no serían una componente del mismo.

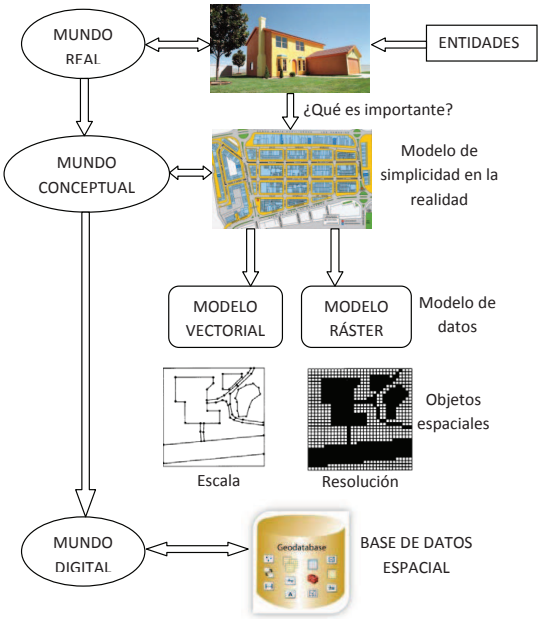

Fig. 15. Modelos conceptuales de datos.

Ahora vamos a detallar lo que se expone en la Figura 15, donde podemos observar que existen dos modelos conceptuales para representar los fenómenos geográficos o entidades del mundo real. Una primera aproximación, la más intuitiva, se basa en considerar la localización y las propiedades espaciales de las entidades. Este modelo, conocido como *modelo vectorial* visible en la Figura 16, utiliza tres tipos de *objetos espaciales* básicos para representar las entidades: *puntos*, *líneas* y *polígonos*. Por ejemplo, un lago (entidad) se representa mediante un polígono (objeto espacial). Los límites de las entidades están representados de forma explícita, por lo que este modelo se aproxima mucho a la cartografía tradicional y resulta más sencillo para cualquiera que haya manejado un mapa.

El concepto clave en el modelo vectorial es la *escala* tal y como se puede observar en la Figura 15. La escala es la relación entre las dimensiones del modelo y del terreno. Puede ser expresada de forma gráfica, mediante una regla graduada unida al mapa, o numérica, a través de un cociente de numerador unidad. Suele haber costumbre de utilizar sólo el denominador (decir, por ejemplo, escala *cien mil* o escala *cinco mil*). La escala disminuye cuando el denominador aumenta, de forma que la escala 1:100.000 es menor que la 1:5.000. La escala, que esté estrechamente relacionada con el concepto

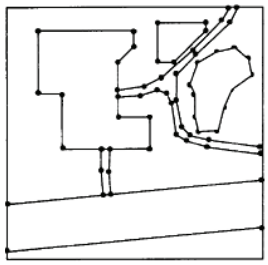

Fig. 16. Modelo vectorial.

de *generalización cartográfica*, tiene implicaciones de largo alcance mucho más allá de una simple relación cuantitativa, tanto para la concepción y elaboración del modelo como para la eficacia de su representación en cualquier tipo de salida gráfica.

Una aproximación totalmente diferente es el *modelo ráster*. En este modelo no se tienen en cuenta la localización ni las propiedades espaciales de las entidades. Consiste en superponer una malla regular sobre el terreno y registrar el valor de una variable determinada para cada una de las celdas de la malla. En consecuencia, en el modelo ráster existe un único objeto espacial: la *celda* o *píxel*. La representación ráster más inmediata e intuitiva es una imagen; el valor que se almacena en cada píxel es un nivel de color o tono. Por ejemplo, un lago (entidad) se representa por un conjunto de celdas (objetos espaciales) contiguas que tienen un mismo valor temático (mismo tono de azul). No se representan las fronteras de las entidades de forma explicita, como en el modelo vectorial, aunque se pueden deducir de forma aproximada a partir de los valores que forman las celdas. Si para el modelo vectorial el concepto clave era la escala, en el caso del modelo ráster el concepto fundamental es la *resolución*. La resolución indica la superficie sobre el terreno representada por cada celda (por ejemplo, 10x10 metros). Cuando más pequeña sea la celda, más detallada será la representación de la realidad. En general, se recomienda adoptar un tamaño de celda cuyo lado sea mitad de la longitud de la entidad más pequeña que se quiera representar.

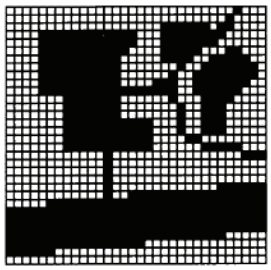

Fig. 17. Modelo ráster.

La georreferenciación de las celdas no es absoluta y directa, como en el caso de los objetos espaciales vectoriales, sino relativa. Se obtiene a partir de su número de fila y columna. La localización absoluta se calcula mediante información adicional: número de filas y de columnas, tamaño de las celdas y coordenadas absolutas de las esquinas de la malla. Otra

diferencia conceptual importante es que en el modelo ráster cubre la totalidad del espacio; no existen, como en el modelo vectorial, *espacios vacíos* entre las entidades individuales seleccionadas.

Una vez expuesta estas dos formas de representar la componente espacial de la información geográfica surge la pregunta sobre cuándo utilizar aproximación vectorial y cuándo usar ráster para construir un modelo de la realidad. Por tanto, de forma genérica el *modelo vectorial* suele ser adecuado para representar entidades artificiales, como carreteras o edificios, cuyas fronteras están bien definidas. Las entidades naturales, sin embargo, se representan mejor mediante el *modelo ráster* porque presentan zonas de transición difusas en lugar de bordes marcados, como ocurre, por ejemplo, con la vegetación. También se representan mejor en el modelo ráster las variables continuas que no están soportadas directamente sobre una entidad, como la temperatura o la precipitación, es decir, variables climáticas.

Ya hemos hablado de objetos espaciales planos (desde las 0 dimensiones del punto a las dos dimensiones del polígono y la celda). Pero existe otra posibilidad: una superficie continua y ondulada en tres dimensiones, cuyos puntos están representados por las coordenadas (x,y) en el espacio ortogonal plano y una tercera coordenada (z) que corresponde a una variable temática continua. Este tipo de construcciones se conocen como *modelos digitales del terreno* (MDT). Un modelo *análogo* podría ser, por ejemplo, una maqueta a escala de una porción de territorio. El MDT más común e intuitivo es el que representa la variable altitud en la coordenada z, denominado *modelo digital del elevaciones*, aunque existen otras posibilidades, como representar la pendiente o la orientación. Esta forma de construir un MDT correspondería a la aproximación ráster, pero también se pueden elaborar según el modelo vectorial; por ejemplo, mediante *curvas de nivel* (isolíneas de igual altitud) o *redes de triángulos irregulares*, más conocidas como estructuras *TIN* (*triangulated irregular networks*). Un concepto fundamental para los dos modelos, vectorial y ráster, es el de *topología*. Las relaciones topológicas indican la posición relativa entre los objetos del modelo. Técnicamente se dice que son aquéllas que permanecen invariables bajo transformaciones afines, como un cambio de escala o la rotación de un mapa. Algunas de las más importantes son la conectividad, la adyacencia, la inclusión y la proximidad. Ambos modelos permiten considerar la topología entre objetos espaciales, aunque lo hacen de forma diferente. En el modelo vectorial, se basa en el tratamiento de los vértices que definen la localización y forma de los objetos espaciales (puntos, líneas y polígonos), mientras que en el ráster se fundamenta en el tratamiento de la matriz de valores almacenados en los objetos espaciales (las celdas de la malla artificial que recubre el espacio).

Aunque el concepto de topología puede parecer muy complejo, lo utilizamos en nuestra vida diaria continuamente; por ejemplo, para indicar una dirección: *cerca de la plaza, junto a la parada del bus*. Esta forma de ubicar un objeto en el espacio se conoce como *posicionamiento relativo*, a diferencia del *absoluto*, que se determina respecto a un sistema de referencia bien definido y se expresa en coordenadas espaciales o proyectadas.

A continuación veremos una aproximación metodológica al proceso de elaboración de un modelo. El primer paso es definir con claridad los objetivos del estudio y las necesidades de información que va a requerir, pero no más. Suele ser conveniente consultar múltiples fuentes, gráficas, documentales y cualquier otro trabajo preliminar que pueda resultar de interés. Una vez establecidos los objetivos y el marco general del proyecto, se inicia la fase de *selección*, cuya primera tarea es definir el espacio geográfico que va a abarcar el modelo. Después, debe decirse cuáles son las entidades relevantes para el propósito del estudio, cuáles las variables temáticas necesarias y los aspectos temporales (momento de la captura, duración del estudio, necesidades de actualización de los datos, etc.). En este punto hay que escoger el modelo de datos más adecuado para cada tipo de información y establecer los procedimientos para combinar y analizar esa información en función de los objetivos que se pretenden alcanzar. Habrá que consultar información espacial disponible y su calidad para determinar si es adecuada para el propósito deseado. Si no lo fuera, sería necesario diseñar y ejecutar las campañas de captura de los datos requeridos, dentro de las limitaciones presupuestarias de todo proyecto. Normalmente es necesario realizar una *clasificación* de la información para agrupar los elementos de características similares y organizar la construcción del modelo, evitando que se vuelva demasiado complejo. Desde el punto de vista más formal, aunque más claro, el proceso de construcción de un modelo también puede contemplarse como la superposición de varios niveles de abstracción. El punto de partida son las propias entidades del mundo real (espacios naturales, campos de cultivo, redes de transportes o áreas urbanas).

El primer nivel de abstracción conceptual o *modelo de datos* (al que corresponden los modelos vectorial y ráster que acabamos de ver), en el que trabajarían los especialistas sectoriales, como arquitectos, ingenieros, geógrafos, geólogos o ecólogos. Estos profesionales elaboran estudios y teorías sobre los procesos que tienen lugar sobre el territorio y deciden cómo modelizar los fenómenos geográficos.

Un segundo nivel de abstracción se ocupa de traducir el modelo conceptual a un modelo lógico que pueda ser implementado en los equipos informáticos. Esto es lo que se denomina *estructura de datos* y de ellos se ocupan los técnicos y analistas SIG. A este nivel también se definen las operaciones y herramientas de análisis espacial.

El tercer y último nivel de abstracción es el denominado *modelo digital*, en el que los ingenieros de software y programadores especializados en SIG se encargan de construir las aplicaciones con las características y funcionalidades definidas en el modelo anterior.

A. Linealización de la información espacial

Sabemos que para poder trabajar con datos espaciales en una red P2P, debemos ser capaces de conservar la localización y direccionamiento de la información a través de estructuras basadas en índices. Con ellos conseguimos resolver problemas tales como: (a) *particionado del espacio y mapeado*, que datos

situados cerca en su espacio nativo puedan ser mapeados en el mismo par o pares que están cercanos en la red superpuesta. (b) *Procesado de consulta*, para el procesado de la consulta se obtienen mejores resultados. (c) *Balanceo de carga*, también conseguimos que la carga de información en cada par sea aproximadamente la misma [Shu et al., 2005].

Pero para poder gestionar los datos espaciales y almacenarlos en un entorno P2P, debemos ir un poco más allá. Hace falta realizar un linealizado de dicha información para pasar de un espacio de datos en 2D a 1D. Para ello, existen muchos métodos (curvas de llenado del espacio) como de los que vamos a hablar a continuación, facilitan esta tarea.

1) Z-order curve: Concretamente, se trata de una función que es capaz de mapear datos multidimensionales a unidimensionales mientras conserva la localización de los puntos de datos [Orenstein and Merrett, 1984]. El valor Z (Z-value) de un punto multidimensional se calcula simplemente entrelazando las representaciones binarias de las coordenadas de sus valores. Una vez los datos son clasificados de esta forma, algunas estructuras de datos unidimensionales se pueden usar para trabajar con ellos, tales como *árboles de búsqueda binaria*, *B-trees*, *listas skips* o *tablas de dispersión*.

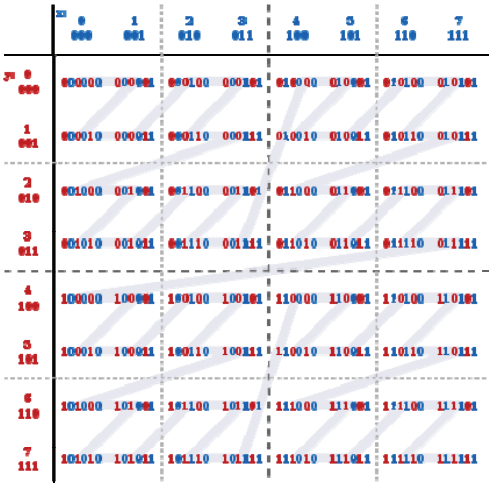

Fig. 18. Z-order curve.

La Figura 18 muestra los Z-valores para el caso de dos dimensiones con coordenadas comprendidas entre $0 \leq x \leq 7$, $0 \leq y \leq 7$. Entrelazando los valores de las coordenadas binarias se obtienen los Z-valores como se muestra. La conexión de los valores Z en su orden numérico produce la curva en forma de Z de manera recursiva. Para trabajar con esta función de linealización en *Java*, la podemos encontrar ya implementada en la clase `MortonList` ubicada en la librería Treemappa[4].

2) Row-major order y column-major order: Estamos ante otra forma de linealizar la información espacial. *Row-major order* (linealización basada en filas) y *column-major order* (linealización basada en columnas) son métodos que permite almacenar información multidimensional en arrays de forma lineal. Partiendo de la estructura estándar de una matriz, las filas definen el primer índice de un array de dos-dimensiones (*Primer índice*, *Segundo índice*) y columnas definen el segundo índice. La estructura de la matriz es compleja para pasar correctamente a array y viceversa. También es importante saber que recorrer un array en el cual los elementos están linealizados, es más rápido que acceder a elementos que no son continuos en memoria lo que hace más útil todavía este método. Vamos a describir como funciona dicho método, a partir de la Figura 19 y sabiendo que esta matriz en un lenguaje de progamación como *Java* sería $intA[8][8] = \{\{0, 2, 3..., 7\}, ..., \{55, 56, 57, ..., 63\}\}$ la linealización aplicando el método *Row-major order* quedaría `1 2 3 4 5 6 ... 64` y los datos con coordenadas $(0, 0)$ y $(1, 1)$ en 2D, se corresponde con 0 y 9 en 1D.

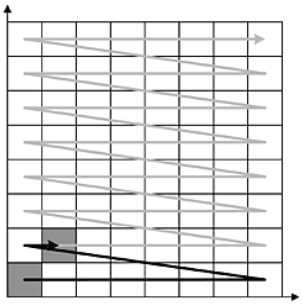

Fig. 19. Row-major order.

3) Otros métodos de linealización espacial: Existen muchas curvas de llenado del espacio [Samet, 2006] que permiten definir funciones que transforman puntos n-dimensionales en 1D y que se pueden aplicar a datos espaciales (2D o 3D) como por ejemplo: *row-prime order*, *Hilbert-order*, *Cantor-diagonal order*, *Spiral order*, *Gray order*, *double gray order*, *U order*

VI. Indexación espacial basada en P2P

A través de la indexación espacial somos capaces de conservar localización y direccionamiento de la información espacial a través de estructuras basadas en índices. Conservar la localización supone registrar qué información espacial vecina se almacena en pares cercanos, mientras que la direccionalidad implica que estructura de índices conservan la orientación. Por tanto, si una estructura de índices conserva estas propiedades entonces buscar en el índice supone una mejora en la evaluación de la consulta. A continuación vamos a describir algunos índices espaciales basados en P2P que aparecen descritos en [Zhang et al., 2011].

P2PR-tree: esta estructura organiza los pares en una superposición de árbol jerárquico, y cada par expone sus datos espaciales contenidos como *peerMBR*. Primero, el conjunto completo es dividido en 4 bloques iguales, y entonces cada bloque es subdividido en 4 bloques iguales. En cada grupo, los pares se comunican entre sí a través del árbol. Si el número de pares en un grupo excede un límite, ese grupo debe ser divido en subgrupos. Así, un árbol distribuido se construye jerárquicamente mediante tales divisiones. Todos los pares se localizan en los nodos hoja, y cada par debe mantener un camino desde él hasta la raíz, que proporcionará información

[4]http://www.soi.city.ac.uk

de navegación durante la búsqueda espacial. Cuando una consulta se emite a un par, el par debe juzgar cual será el par destino más aproximado y dejar el mensaje en el siguiente punto, la consulta se direcciona desde abajo hasta la raíz y entonces desde la raíz al destinatario. El principal inconveniente de los P2PR-tree es que este no es un árbol balanceado, por eso si los datos se separan, algunos pares deben mantener caminos de información extremadamente largos, que harán las búsquedas bajen su rendimiento. Veamos algunas características de este método de indexación espacial basado en P2P:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* Quadtree + R-tree (static + dynamic)
- *Topología:* árbol.
- *Tipo de dato:* datos espaciales.
- *Método de consulta:* encaminamiento a lo largo del camino $hoja \rightarrow raiz \rightarrow hoja$.
- *Coste de la consulta:* $O(logN)$.
- *Coste de añadir/eliminar pares:* $O((logN)^2)$.
- *Insertar/eliminar datos:* $O(logN)$.
- *Balanceo de carga:* no dispone.
- *Estrategia de actualización:* no dispone.

NR-tree: es una versión distribuida del R*-tree. Los pares se clasifican en pares pasivos y súper-pares. Un súper-par gestiona un número de pares pasivos, y así ellos forman un grupo. Si el número de pares pasivos excede un número se dividen. Los súper-pares son los encargados de formar los R*-tree distribuidos. Veamos algunas características de NR-tree:

- *Tipo de consulta:* consulta de ventana, consulta de los k vecinos más cercanos (kNN).
- *Partición del espacio:* partición como en KD-tree.
- *Topología:* como en CAN + árbol.
- *Tipo de dato:* datos espaciales.
- *Método de consulta:* protocolo de búsqueda R*-tree distribuido entre súper-pares y par, y super-pares que usan CAN para comunicarse con otros.
- *Coste de la consulta:* $O(logN + dN^{1/d})$ donde d es la dimensionalidad del espacio.
- *Coste de añadir/eliminar pares:* $O(logN + 2d)$.
- *Insertar/eliminar datos:* $O(logN + dN^{1/d})$.
- *Balanceo de carga:* no dispone.
- *Estrategia de actualización:* promoción súper-par.

P2PRdNN-Tree: está bajo el contexto de la consulta del vecino más cercano inversa (Reverse Nearest Neighbor Query, RNNQ). Como el NR-tree, el P2PRdNN-Tree es una topología P2P basada en súper-pares. Un pequeño subconjunto de pares con relativamente alta estabilidad y capacidad de computación se seleccionan como súper-pares. La diferencia con el NR-tree, es que los súper-pares en el P2PRdNN-Tree usan un canal principal para entregar mensajes, que es una manera de transmitir. Veamos algunas características de este método de indexación espacial basada en P2P:

- *Tipo de consulta:* consulta RNN.
- *Partición del espacio:* regiones de superposición.
- *Topología:* árbol.
- *Tipo de dato:* datos espaciales.
- *Método de consulta:* canal principal + dominio de búsqueda usando árbol.
- *Coste de la consulta:* $O(logN)$
- *Coste de añadir/eliminar pares:* no dispone
- *Insertar/eliminar datos:* no dispone.
- *Balanceo de carga:* no dispone.
- *Estrategia de actualización:* no dispone.

MAAN: MAAN (Multi-Attribute Addressable) [Cai et al., 2004] se centra en consultas de rango multi-atributo en sistemas P2P. Usan dispersión local para valores de atributos en *Chord*. Esta propuesta usa un mapeo directo del dominio de datos para el espacio de *Chord* o asume que el rango de datos de entrada y la distribución se conoce de antemano para crear un mapeo balanceado. Veamos algunas características de este método:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* punto de dispersión para pares.
- *Topología:* anillo.
- *Tipo de dato:* puntos.
- *Método de consulta:* similar a *Chord*, pero para una consulta $d-dimensional$ usara el mecanismo de búsqueda como en *Chord* durante d veces.
- *Coste de la consulta:* $O(dlogN)$.
- *Coste de añadir/eliminar pares:* $O(dlogN^2)$
- *Insertar/eliminar datos:* $O(dlogN)$.
- *Balanceo de carga:* no dispone.
- *Estrategia de actualización:* no dispone.

Mercury: [Bharambe et al., 2004] es un sistema de consulta en rango multi-atributo similar a MAAN. Usa múltiples círculos superpuestos y organiza los pares del sistema en estas superposiciones. Se puede acceder a cada círculo a través de un atributo que es ordenado mediante el valor de lo que se conoce como eje. Dos ejes están conectados a través de un link construido por los pares localizados en dicho eje. Veamos algunas como es Mercury:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* punto distribuido para pares.
- *Topología:* múltiples anillos.
- *Tipo de dato:* puntos.
- *Método de consulta:* similar a *Chord*, a diferencia de que, Mercury usa histograma para decidir que dimensiones deberían ser consultadas primero.
- *Coste de la consulta:* $O(dlogN)$.
- *Coste de añadir/eliminar pares:* $O(d(logN)^2)$
- *Insertar/eliminar datos:* $O(dlogN)$.
- *Balanceo de carga:* histograma basado en carga balanceada.
- *Estrategia de actualización:* no dispone.

PRoBe: [Sahin et al., 2005] organiza los pares en un espacio lógico, que es similar a CAN. La dimensionalidad de este espacio se asigna al número de atributos del rango. Cada dimensión corresponde a un atributo y es acotado por el dominio del valor del atributo correspondiente. A cada par del sistema se le asigna una zona, procedente de la división de regiones, y es responsable de mantener los datos mapeados en esa zona. Veamos algunas propiedades del método de indexación espacial PRoBe:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* división de la carga de forma equitativa.
- *Topología:* CAN.
- *Tipo de dato:* puntos y datos espaciales.
- *Método de consulta:* mecanismo de consulta igual a CAN, a diferencia de que PRoBe usa caché para hacer las consultas más rápidas.
- *Coste de la consulta:* $O(dN^{1/d})$.
- *Coste de añadir/eliminar pares:* $O(2d)$.
- *Insertar/eliminar datos:* $O(dN^{1/d})$.
- *Balanceo de carga:* modo virtual.
- *Estrategia de actualización:* no dispone.

BATON: (Jagadosj et al., 2005) es una estructura basada en superposición en un árbol binario balanceado en el que cada par de la red mantiene un par del árbol. Un par puede conectar a otros pares a través de 4 tipos diferentes de links: $linkhacialospadres$ que apuntan a los pares padres, $linkhacialoshijos$ que apuntan hacia pares hijos, $linkadyacentes$ apuntando a pares adyacentes que mantienen rangos adyacentes de valores, y $linkvecinos$ apuntando a pares vecinos seleccionados dentro del mismo nivel y tienen una distancia igual a potencia de dos desde el par. En BATON, cada par en el árbol, tanto pares hojas como pares internos, se les asigna un rango de valores. En cada rango de valores directamente gestionado por un par es mayor que el rango de valores gestionado por sus pares adyacentes izquierdos, mientras es menor que el rango de valores gestionados por sus pares adyacentes derechos. Cuando un par recibe una solicitud de consulta, si el valor buscado no cae en su propio rango de valores, la respuesta es reenviada a (1) un par situado a su izquierda en la tabla de rutas cuyo límite superior es todavía mayor que el valor buscado o a (2) un par situado a la derecha en su tabla de rutas cuyo límite inferior sea todavía menor que el valor de búsqueda si tal par existe, o sino, (3) la solicitud de consulta es redirigida a otro de sus hijos izquierdo o derecho de los pares adyacentes. Para pares eliminados del árbol o balanceados, BATON debería ser reestructurado como muestra la Figura 20. Veamos como es BATON según sus características:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* puntos de dispersión para pares, curva de llenado del espacio.
- *Topología:* árbol.
- *Tipo de dato:* puntos.
- *Método de consulta:* mecanismo de búsqueda en árbol usando vecinos, padres e hijos.
- *Coste de la consulta:* $O(log_m N)$.
- *Coste de añadir/eliminar pares:* $O(mlog_m N)$.
- *Insertar/eliminar datos:* $O(log_m N)$.
- *Balanceo de carga:* movimiento de la carga y los pares.
- *Estrategia de actualización:* no dispone.

RT-CAN: [Wang et al., 2010] el campo de investigación de RT-CAN es el *cloud computing* que es una nueva área para la indexación espacial en P2P. RT-CAN combina CAN y R-tree para estudiar los problemas de indexación espacial en sistemas cloud. A cada par en el sistema se le asignan

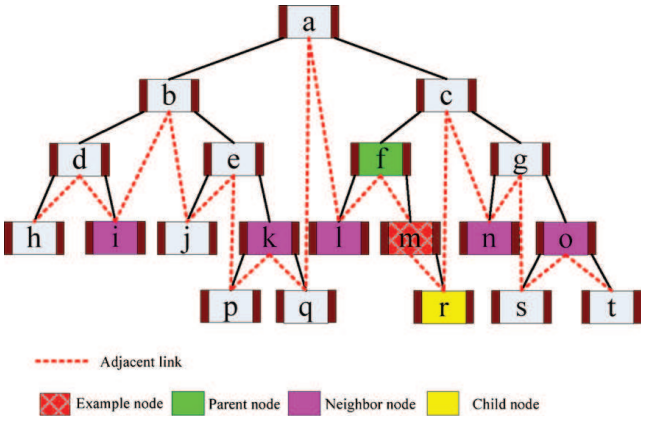

Fig. 20. Estructura BATON.

2 roles: par de almacenamiento y par de superposición. Los pares de almacenamiento mantienen una porción de los datos de aplicación y construyen un R-tree para sus datos locales. El par de superposición es un par en la estructura de superposición CAN, y es el responsable de una partición de CAN. El par de almacenamiento selecciona los pares situados en el nivel superior de las hojas del R-tree cuando quiere informar a CAN. Para llevar a cabo este proceso de información este par de almacenamiento se pone en contacto con un par superpuesto. RT-CAN adopta una estrategia de propagación de información, en la cual si el radio del par N del R-tree que se va a encargar de informar es menor que un umbral, entonces N envía la información al grupo de pares situados en el centro de N; sino a lo envía todos los pares de CAN superpuestos con N en su índice global. Para una consulta de ventana, primero, la consulta se redirecciona al par de CAN C donde la zona que contiene este par es el centro de la ventana de consulta; entonces la consulta es recursivamente redirigida a todos los vecinos que se solapan con la ventana de consulta; para recibir la consulta, el par de almacenamiento busca su índice local y devuelve los resultados. Veamos algunas propiedades de este método de indexación espacial:

- *Tipo de consulta:* consulta de ventana, consulta kNN.
- *Partición del espacio:* partición como en CAN.
- *Topología:* CAN.
- *Tipo de dato:* puntos y datos espaciales.
- *Método de consulta:* mecanismo de consulta como en CAN.
- *Coste de la consulta:* $O(logN)$.
- *Coste de añadir/eliminar pares:* $O(logN)$.
- *Insertar/eliminar datos:* $O(logN)$.
- *Balanceo de carga:* no dispone.
- *Estrategia de actualización:* acorde al radio de consultas frecuentes para actualizar frecuentemente, ajustando el nivel de los pares par ser publicado.

DHR-tree: [Wei and Sezaki, 2006] combina Hilbert R-tree [Kamel and Faloutsos, 1993] y P-tree [Crainiceanu et al., 2004] que es un sistema P2P superpuesto de versión distribuida de B^+-tree. Cada par se encarga del control de algún objeto espacial compuesto por un MBR en

el espacio. Veamos algunas características de DHR-tree:

- *Tipo de consulta:* consulta de rango.
- *Partición del espacio:* superposición de regiones.
- *Topología:* P-tree.
- *Tipo de dato:* puntos y datos espaciales.
- *Método de consulta:* usando curva de Hilbert y mecanismo de consulta P-tree.
- *Coste de la consulta:* $O(logN)$.
- *Coste de añadir/eliminar pares:* $O(logN)$.
- *Insertar/eliminar datos:* $O(logN)$.
- *Balanceo de carga:* no dispone.
- *Estrategia de actualización:* no dispone.

Distributed Quadtree: [Tanin et al., 2007] usa MX-CIF quadtree para dividir el espacio. La función de división es para marcar cada objeto espacial con varios puntos de control. Una función de distribución consistente [Eastlake and Jones, 2001] puede ser usada para mapear estos puntos de control en valores de una dimensión que pueden ser indexados por *Chord*. Así, con estos puntos de control, un objeto espacial o una consulta espacial pueden ser indexados o buscados por pares en *Chord*. Hay que tener en cuenta que, un objeto espacial puede ser indexado por más de un par en *Chord*, y esto podría incrementar el coste de mantenimiento. Comprobemos los atributos de este método de indexación espacial:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* partición quadtree.
- *Topología: Chord*.
- *Tipo de dato:* datos espaciales.
- *Método de consulta:* usando funciones de dispersión para *Chord*, a través de mecanismos de consulta *Chord* para recuperar objetos.
- *Coste de la consulta:* $O(logN)$.
- *Coste de añadir/eliminar pares:* $O((logN)^2)$.
- *Insertar/eliminar datos:* $O(logN)$.
- *Balanceo de carga:* no dispone.
- *Estrategia de actualización:* no dispone.

Squid: [Schmidt and Parashar, 2003] usa la curva de llenado del espacio de Hilbert para mapear el espacio multidimensional en un espacio de dimensión uno que puedan ser indexados por *Chord*. Las operaciones de combinación de pares y eliminación de pares de la red son las mismas que con *Chord*. Comprobemos algunas de las características de Squid:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* curva de Hilbert.
- *Topología: Chord*.
- *Tipo de dato:* puntos.
- *Método de consulta:* usa la curva de Hilbert para mapear consultas espaciales en 1D del espacio, y entonces usa mecanismos de consulta *Chord*.
- *Coste de la consulta:* $O(logN)$.
- *Coste de añadir/eliminar pares:* $O((logN)^2)$.
- *Insertar/eliminar datos:* $O(logN)$.
- *Balanceo de carga:* balanceo de carga cuando se juntan pares en tiempo de ejecución.
- *Estrategia de actualización:* no dispone.

SCRAP: [Ganesan et al., 2004] combina SkipGraph y curvas de llenado del espacio. En SCRAP y SkipIndex es común que ellos codifiquen los objetos espaciales con claves en una dimensión y asociar las claves con los pares en SkipGraph para que el mantenimiento distribuido y las consultas generales en la indexación espacial pueda ser implementada a través de SkipGraph. Veamos a continuación algunas propiedades de SCRAP:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* curva de Hilbert.
- *Topología:* SkipGraph.
- *Tipo de dato:* puntos.
- *Método de consulta:* usando la curva de Hilbert para mapear consultas espaciales en 1D, y entonces usar mecanismos de consulta del SkipGraph.
- *Coste de la consulta:* $O(logN)$.
- *Coste de añadir/eliminar pares:* $O(logN)$.
- *Insertar/eliminar datos:* $O(logN)$.
- *Balanceo de carga:* movimiento de carga y pares.
- *Estrategia de actualización:* no dispone.

Z-NET: [Ganesan et al., 2004] Z-NET y SCRAP son lo mismo practicamente; la única diferencia es que, en SCRAP los autores proponen el uso de curvas de llenado de espacio (Z-curve o curva de Hilbert), y en Z-NET los autores especifican el uso exclusivo de Z-curve. Veamos a continuación algunas propiedades de Z-NET:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* Z-curve.
- *Topología:* SkipGraph.
- *Tipo de dato:* punto de datos.
- *Método de consulta:* similar a SCRAP.
- *Coste de la consulta:* $O(logN)$.
- *Coste de añadir/eliminar pares:* $O(logN)$.
- *Insertar/eliminar datos:* $O(logN)$.
- *Balanceo de carga:* balanceo de carga al agregar pares en tiempo de ejecución.
- *Estrategia de actualización:* no dispone.

SPATIALP2P: [Kantere et al., 2009] estudia como almacenar e indexar datos espaciales en sistemas P2P. SPATIALP2P mantiene que la curva de llenado del espacio no mantiene lo suficientemente bien la direccionalidad y localidad, por eso adopta un método en el que los pares que almacenan información cercana en el espacio sean cercanos ellos también dentro de la red P2P. SPATIALP2P solo se contempla el caso de espacio en 2 dimensiones, en el que el espacio es igualmente dividido en celdas y a cada celda se le asigna una coordenada en 2 dimensiones. SPATIALP2P propone una distancia métrica entre 2 celdas cualquiera. La métrica es una tupla binaria en el que el primer item es el que contiene la mayor diferencia de distancia entre las 2 celdas a partir de las coordenadas x e y, y el segundo item es el elemento más cercano. Dadas dos distancias d_1 y d_2, si el primer elemento en d_1 es mayor que el d_2, entonces d_1 es mayor que d_2; si el primer item es igual, el orden de d_1 y d_2 depende del segundo item. Acorde a la métrica de distancias, a una celda se le asigna el par más cercano para gestionarlo. Además de que, cada par p no esta solo ligado con 4 sucesores respectivamente, sino que también está conectado a algunos pares indexados que a través de link largos. Esta forma de conectarse a otros pares es similar a la

que sigue *Chord* con su tabla de apuntadores. Comentemos ahora algunas propiedades de SPATIALP2P:

- *Tipo de consulta:* consulta de ventana.
- *Partición del espacio:* partición en grid.
- *Topología:* CAN + long links.
- *Tipo de dato:* puntos.
- *Método de consulta:* uso de vecinos y de long links.
- *Coste de la consulta:* $O(logN)$.
- *Coste de añadir/eliminar pares:* no dispone.
- *Insertar/eliminar datos:* no dispone.
- *Balanceo de carga:* no dispone.
- *Estrategia de actualización:* no dispone.

A continuación, vamos a sintetizar y clasificar en la Tabla I un resumen con los índices centralizados principales, sistemas P2P y la combinación de índices centralizados y sistemas P2P.

VII. Herramientas para el estudio de redes P2P

Las redes P2P son potencialmente enormes ya que suelen estar compuestas de millones de pares. En estos entornos continuamente hay pares entrando y saliendo del sistema, por lo que evaluar los protocolos en un entorno real no es una tarea fácil. Por lo que, existen una serie de herramientas que permiten el estudio, la validación y la implementación de sistemas basados en redes P2P así como de nuevas redes P2P de forma relativamente rápida. Por un lado, tenemos los simuladores que nos permiten testear nuestras propuestas y por otro las librerías que implementan ciertas redes ya existentes.

A. Simuladores

Un motor de simulación es una aplicación en la cual se pueden llevar a cabo simulaciones y obtener a partir de ellas un conjunto de resultados que nos permitan realizar análisis sobre los mismos. El motor también se encarga del tiempo de simulación del experimento además de las interacciones que se producen entre los elementos [Pérez-Miguel et al., 2009].

OverSim: descrito en [Baumgart et al., 2007], OverSim es un simulador de eventos basado en OMNeT++ que permite simular cualquier red P2P, estructuradas o no, y que implementa varios protocolos P2P, entre ellos *Chord* y *Kademlia*. Es fácilmente ampliable mediante módulos programados en C++ y permite la simulación de redes P2P con diversos grados de detalle en función de qué módulo se use para simular la red subyacente.

P2PSim: propuesto en [Barbosa et al., 2007], es un simulador de eventos a nivel de paquete que puede simular sólo superposiciones estructuradas. Posee implementaciones de *Chord*, *Accordion*, *Koorde*, *Kelips*, *Tapestry* y *Kademlia*. La API en C++ está pobremente documentada pero existen clases ejemplo para facilitar la implementación de nuevos protocolos extendiéndolas. Actualmente su desarrollo está abandonado.

PlanetSim: [García et al., 2005] simulador por eventos desarrollado en *Java*. Permite simular redes estructuradas y no estructuradas. Además incluye la implementación de *Chord* y Symphony así como también implementa la Common API. Por desgracia no permite la obtención de estadísticas, con lo que su uso es más bien anecdótico.

PeerSE: [Bischofs and Hasselbring, 2009] simulador desarrollado en *Java*. Permite la simulación tanto del overlay como del nivel físico.

PeerSim[5]**:** es un simulador libre desarrollado en el departamento de ciencias de la computación de la Universidad de Bolonia y actualmente está mantenido por *Mark Jelasity*, *Gian Paolo Jesi*, *Alberto Montresor* y *Spyros Voulgaris* todos procedentes de esta Universidad. *PeerSim* ha sido elegido para realizar este trabajo ya que esta entre los simuladores más conocidos entre la comunidad investigadora. Además, permite la simulación de redes P2P estructuradas y no estructuradas con hasta un millón de pares. Se ha implementado en *Java* y publicado bajo la licencia GPL dentro del proyecto BISON. Es ampliable mediante módulos por lo que es altamente escalable, extremadamente configurable y muy flexible. Todo esto hace que sea perfecto para el trabajo que se ha llevado a cabo. Pero también, el motor de simulación está construido a base de componentes a los que se pueden añadir *plugging*, ya que el objetivo es re-usar módulos existentes. Estos módulos pueden ser de diferentes tipos, por ejemplo hay módulos que pueden construir e inicializar la red interna, módulos que pueden manejar diferentes protocolos, módulos para controlar y modificar la red, por lo que es capaz de facilitar la codificación de nuevas aplicaciones. Dentro de *PeerSim* hay dos tipos de simulaciones, por un lado está la simulación basada en eventos y por otro la simulación basada en ciclos.

- *Simulación basada en ciclos*, es la simulación más fácil y clara ya que no se producen mensajes entre los pares, no existe transporte y la simulación es síncrona. El control se asigna a cada par de manera cíclica, para procesar así sus operaciones. Esta simulación es específica para protocolos epidémicos que son aquellos que hacen uso de algoritmos epidémicos para la distribución de información [Lavastida-López and Almeida-Cruz, 2009].
- *Simulación basada en eventos*, es una simulación más realista basada en mensajes que representan el transporte. En este modelo de simulación, cuando se genera un mensaje se envía al protocolo concreto de un par, para ello hay un método que se encarga de gestionar los mensajes entrantes y manejarlos. Puede ser usado para protocolos epidémicos y normales.

Como se puede ver en la Figura 21, el motor de simulación (Simulation Engine) tiene una red (Network) como variable global formada por un conjunto de pares (Node) representados en un array, que se pueden ver como cuadrados blancos en el diagrama. A los pares se le asignan, en forma de pila, los protocolos de comunicación dentro de la red. Estos protocolos van a definir la forma en que los pares deben de comunicarse entre sí y cómo deben de comportarse ellos mismos dentro de la red. Dentro del motor de simulación tenemos elementos como `init.traces` o `init.random` que se encargan de llevar a cabo tareas de inicialización en la red simulada, tareas tales como la iniciar la topología y el estado de los pares. `control.observe-tman`, `control.observe-slice` se encargan de llevar la observación y modificación de propiedades de la red simulada.

[5] http://peersim.sourceforge.net/

TABLE I. TABLA RESUMEN DE EXTENSIÓN DE ÍNDICES CENTRALIZADOS, SISTEMAS P2P Y COMBINACIÓN DE AMBOS.

Extensión de índices centralizados	Extensión de sistemas P2P	Combinación de índices centralizados y sistemas P2P
P2PR-tree (R-tree)	MAAN (*Chord*)	RT-CAN (R-tree y CAN)
NR-tree (R*-tree)	Mercury (*Chord*)	DHR-tree (Hilbert R-tree y P-tree)
P2PRdNN-tree (R*-tree)	PRoBe (CAN)	Distributed Quadtree (MX-CIF Quadtree y *Chord*)
	BATON (BATON)	Squid (curva de Hilbert y *Chord*)
		SCRAP (curva de Hilbert y SkipGraph)
		Z-NET (Z-curve y SkipGraph)

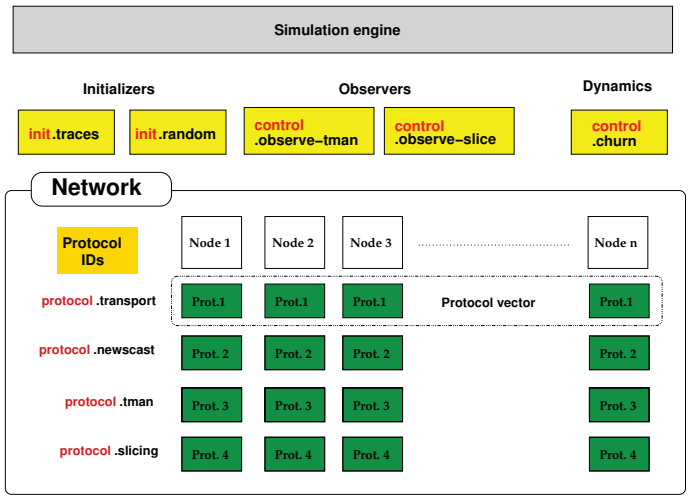

Fig. 21. Estructura del simulador *PeerSim*.

Por otro lado, `control.churn` se ejecuta periodicamente preocupandose por añadir, eliminar y re-iniciar los pares que constituyen la red.

Una vez todos los componentes han sido implementados, la simulación completa debe ser configurada declarando qué componentes se usan y definir de que forma deben interaccionar los componentes entre sí. En *PeerSim*, la simulación se define a través de un fichero de configuración en formato ASCII que ayuda a reducir la sobrecarga. Los ficheros de configuración se dividen en cuatro partes tal y como se puede ver en la Figura 22:

1) *Pasos generales:* donde se especifica el número de pares en la red o los retardos que deben de producirse en la simulación.
2) *Definición del protocolo:* donde se detalla que protocolos van a estar en la pila de protocolos de los pares.
3) *Inicialización:* apartado en el cual se define el estado inicial de la red.
4) *Control de definición:* donde se expone cómo debe iniciarse el protocolo.

Además el simulador *PeerSim* incluye librerías de protocolos ya implementadas tales como *Chord*, *Pastry* y *Kademlia* entre otros. Según se observa en la Figura 23 tenemos el protocolo *Kademlia* simulado con *PeerSim* utilizando como IDE Eclipse. En la parte izquierda de la imagen podemos ver las clases que forman dicho protocolo en el simulador, en el recuadro de arriba a la derecha el fichero de configuración del protocolo y

```
################### general ==================
# network size
SIZE 1000

# parameters of periodic execution
CYCLES 100
CYCLE SIZE*10000

# parameters of message transfer
MINDELAY 0
MAXDELAY 0
...

################### protocols =================
protocol.link peersim.core.IdleProtocol
protocol.avg.step CYCLE
protocol.avg.transport tr

protocol.urt.mindelay (CYCLE*MINDELAY)/100
protocol.urt.maxdelay (CYCLE*MAXDELAY)/100
...

################### initialization ============
init.vals LinearDistribution
init.vals.max SIZE
init.vals.min 1

init.sch CDScheduler
init.sch.randstart
...

################ control =====================
control.0 SingleValueObserver
control.0.step CYCLE
...
```

Fig. 22. Fichero de configuración.

abajo a la derecha la salida por consola como resultado de la simulación.

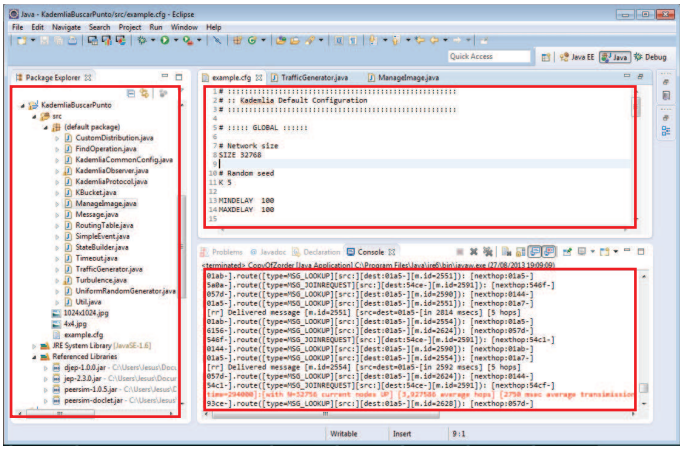

Fig. 23. Protocolo *Kademlia*, simulado con *PeerSim* y ejecutado en Eclipse.

VIII. Implementación de *Chord*, *Kademlia* y *Pastry* en *Peersim*

Como hemos mencionado en la Sección VII-A *PeerSim* tiene algunas librerías donde ya están implementados algunos protocolos. Para el desarrollo de este trabajo hemos utilizado los protocolos *Chord*, *Pastry* y *Kademlia*. A continuación, vamos a describir como están implementados estos protocolos en *PeerSim*.

A. Chord en PeerSim

Para el desarrollo de *Chord* en *PeerSim* se han implementado nueve clases sobre las que vamos a proceder a explicar cual es su función:

- `ChordInitializer:` clase que implementa la interfaz `NodeInitializer`. Se encarga de inicializar todos los pares de la red dando valores a la lista de sucesores del par y generando su tabla de apuntadores.
- `ChordProtocol:` aquí se define el protocolo *Chord*, implementando la interfaz `EDProtocol`. Alberga todos los métodos relacionados con el procesado de eventos sobre los pares. Este protocolo que se agrega a la pila de protocolos que definen un par, se va a encargar de comprobar si el evento recibido es para él o por el contrario debe de redireccionarlo a algún sucesor o par de su tabla de apuntadores. También se encarga de realizar actualizaciones sobre la lista de sucesores y la tabla de apuntadores.
- `CreateNw:` clase utilizada en `ChordInitializer` y que implementa la interfaz `Control`. En ella se encuentran los métodos que definen la lista de sucesores y la tabla de apuntadores de la red para un par.
- `FinalMessage:` clase que define un mensaje que se envía a través de la red e implementa la interfaz `ChordMessage`. Contiene un atributo que guarda el número de saltos que dio el mensaje en la red hasta encontrar su destino.
- `LookUpMessage:` clase que define el mensaje de búsqueda implementando la interfaz `ChordMessage`. Ésta guarda el par emisor del mensaje, el identificador del par objetivo y el número de saltos que da en la red hasta encontrar su destino.
- `MessageCounterObserver:` es la clase que se encarga de visualizar el estado de los mensajes accediendo al número de saltos que dieron los mensajes en la red y calculando media, máximo y mínimo número de saltos de un mensaje. Implementa la interfaz `Control`.
- `NodeComparator:` clase que compara entre si dos pares. Implementa la interfaz `Comparator`.
- `Parameters:` clase encargada de definir los parámetros de envío.
- `TrafficGenerator:` clase que se encarga de crear el tráfico en la red e implementa la interfaz `Control`. Contiene un método que genera todo el tráfico, definiendo el par emisor y receptor de mensajes. Además lanza la simulación introduciendo un retardo de diez milisegundo entre cada emisión de mensaje entre pares.

Un diagrama de clases que pueda mostrar las relaciones entre las clases previamente descritas está en la Figura 24. En él vemos como cada clase se relaciona dentro del protocolo *Chord* implementado en *PeerSim*. En dicho diagrama, tenemos también visualizada la librería *PeerSim* en forma de directorio, la cuál, utiliza el resto de clases para lanzar la simulación.

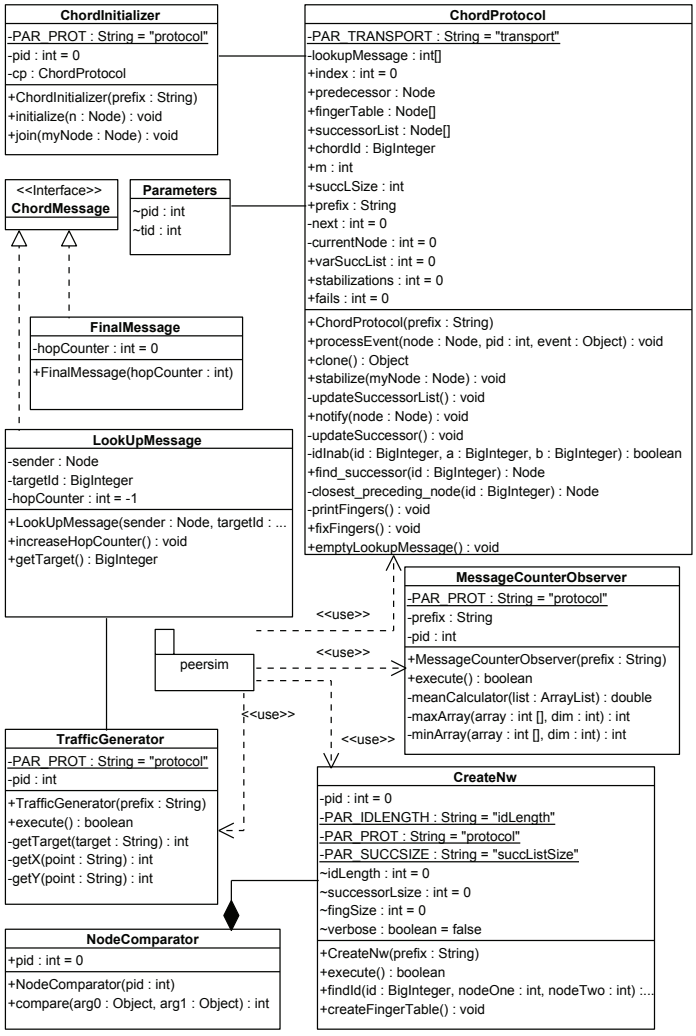

Fig. 24. Diagrama de clases del protocolo *Chord* en *PeerSim*.

B. Kademlia en PeerSim

Para el desarrollo de *Kademlia* en *PeerSim*, en este caso, se ha trabajado sobre quince clases de las que se van a describir brevemente a continuación:

- `CustomDistribution:` clase que implementa la interfaz `Control`. Se encarga de supervisar los pares de la red y asignarles un identificador generado de forma aleatoria dentro del espacio de claves que maneja la red.
- `FindOperation:` clase que representa una operación de búsqueda y ofrece los métodos necesarios para mantener y actualizar el conjunto de pares más cercanos

al par en cuestión. También gestiona el número de solicitudes paralelas que puede controlar el par que en este caso es concretamente un número máximo α.

- `KademliaCommonConfig:` clase que se encarga de establecer los parámetros del protocolo *Kademlia* en la red. El sistema P2P basado en *Kademlia* tiene valores por defecto que sólo pueden ser configurados al inicio de la red.
- `KademliaObserver:` clase que implementa un observador simple basado en tiempo de búsqueda y a través del que calcula la media de saltos encontrados en un nodo de la red.
- `KademliaProtocol:` clase que gestiona todo el protocolo *Kademlia* dentro de la simulación en *PeerSim*.
- `KBucket:` clase que implementa la interfaz `KBucket`. En esta clase se define el *k-bucket* de un par. En *Kademlia* los pares almacenan las referencias a otros pares empleando una especie de lista, que los autores denominaron *k-bucket*.
- `Message:` clase que hereda de la clase `SimpleEvent`. Se encarga de definir como debe de ser un mensaje dentro de la simulación para el protocolo *Kademlia*.
- `RoutingTable:` clase que implementa la interfaz `RoutingTable`. Esta clase hace uso de la clase `KBucket` pues en `RoutingTable` se gestionan los vecinos de un par por lo que hay que modifica la lista *k-bucket*.
- `SimpleEvent:` es una clase que se encarga de almacenar las propiedades pertinentes que debe de contener un evento.
- `StateBuilder:` clase que se encarga de inicializar el llenado de los *k-buckets* para todos los pares de la red. Concretamente cada nodo se añade a la tabla de rutas de otros pares en la red.
- `Timeout:` clase que controla el *timeout* de un evento.
- `TrafficGenerator:` clase que implementa la interfaz `Contol`. Se encarga de crear el tráfico en la red y para ello contiene un método que genera todo el tráfico, en el cual se define el par emisor y el par receptor de mensajes. Una vez definidos los pares lanza la simulación.
- `Turbulence:` clase cuyo único propósito es realizar testeos y crear estadísticas. Para ello, se encarga de añadir y eliminar pares de la red según cierta probabilidad.
- `UniformRandomGenerator:` esta clase se encarga de asignar a los pares un identificador.
- `Util:` clase que contiene algunas utilidades y funciones matemáticas para trabajar con números `BigInteger` y `String`.

Hay disponible un diagrama de clases en la Figura 25 que pueda mostrar las relaciones entre las clases previamente descritas. En él vemos como cada clase se relaciona dentro del protocolo *Kademlia* implementado en *PeerSim*. En dicho diagrama, tenemos también visualizada la librería *PeerSim* en forma de directorio, la cuál, utiliza el resto de clases para lanzar la simulación.

C. Pastry en PeerSim

Para el desarrollo de *Pastry* en *PeerSim*, se crearon doce clases para poder lanzar su simulación. Veamos que hace cada una de ellas:

- `CustomDistribution:` clase que implementa la interfaz `Control`. Se encarga de inicializar la red completa asignándole un identificador único generado aleatoriamente a cada nodo de la red.
- `LeafSet:` clase encargada de encapsular funcionalidades para trabajar con la tabla de los pares hoja dentro de un par *Pastry*, además de añadir *inteligencia* automática y facilitando la extracción de información.
- `Message:` clase que se encarga de definir como debe de ser un mensaje dentro de la simulación para el protocolo *Pastry*.
- `MSPastryCommonConfig:` se encarga de inicializar los parámetros de funcionamiento de la red *Pastry*. Estos parámetros se inicializan solo al comienzo de la simulación.
- `MSPastryObserver:` clase que implementa un observador simple, a partir del cual consulta el estado actual de la red durante la simulación.
- `MSPastryProtocol:` clase que gestiona todo el protocolo *Pastry* dentro de la simulación en *PeerSim*.
- `RoutingTable:` clase que implementa la interfaz `Clonable`. En esta clase se implementan la tabla de rutas para poder redireccionar los mensajes a los pares correspondientes cuando se realiza una búsqueda.
- `StateBuilder:` clase que se usa en `RoutingTable` para poder generar la tabla de rutas de cada par.
- `TrafficGenerator:` clase que implementa la interfaz `Contol`. Se encarga de crear el tráfico en la red y para ello contiene un método que genera todo el tráfico, en el cual se define el par emisor y receptor de mensajes. Una vez definidos los pares lanza la simulación.
- `Turbulence:` clase cuyo único propósito es realizar testeos y crear estadísticas. Para ello, se encarga de añadir y eliminar pares de la red según cierta probabilidad.
- `UniformRandomGenerator:` esta clase se encarga de asignar a los pares un identificador.
- `Util:` clase que contiene algunas utilidades y funciones matemáticas para trabajar con números `BigInteger` y `String`.

Tenemos un diagrama de clases en la Figura 26 que muestra las relaciones entre las clases previamente descritas. En él vemos como cada clase se relaciona dentro del protocolo *Pastry* implementado en *PeerSim*. En dicho diagrama, tenemos también visualizada la librería *PeerSim* en forma de directorio, la cuál, utiliza el resto de clases para lanzar la simulación.

IX. Resultados experimentales

Con el objetivo de trabajar con información espacial en los sistemas P2P, se han realizado varios experimentos en los cuales hemos simulado los protocolos *Pastry*, *Chord* y *Kademlia* en el entorno *PeerSim*. Para poder trabajar con información

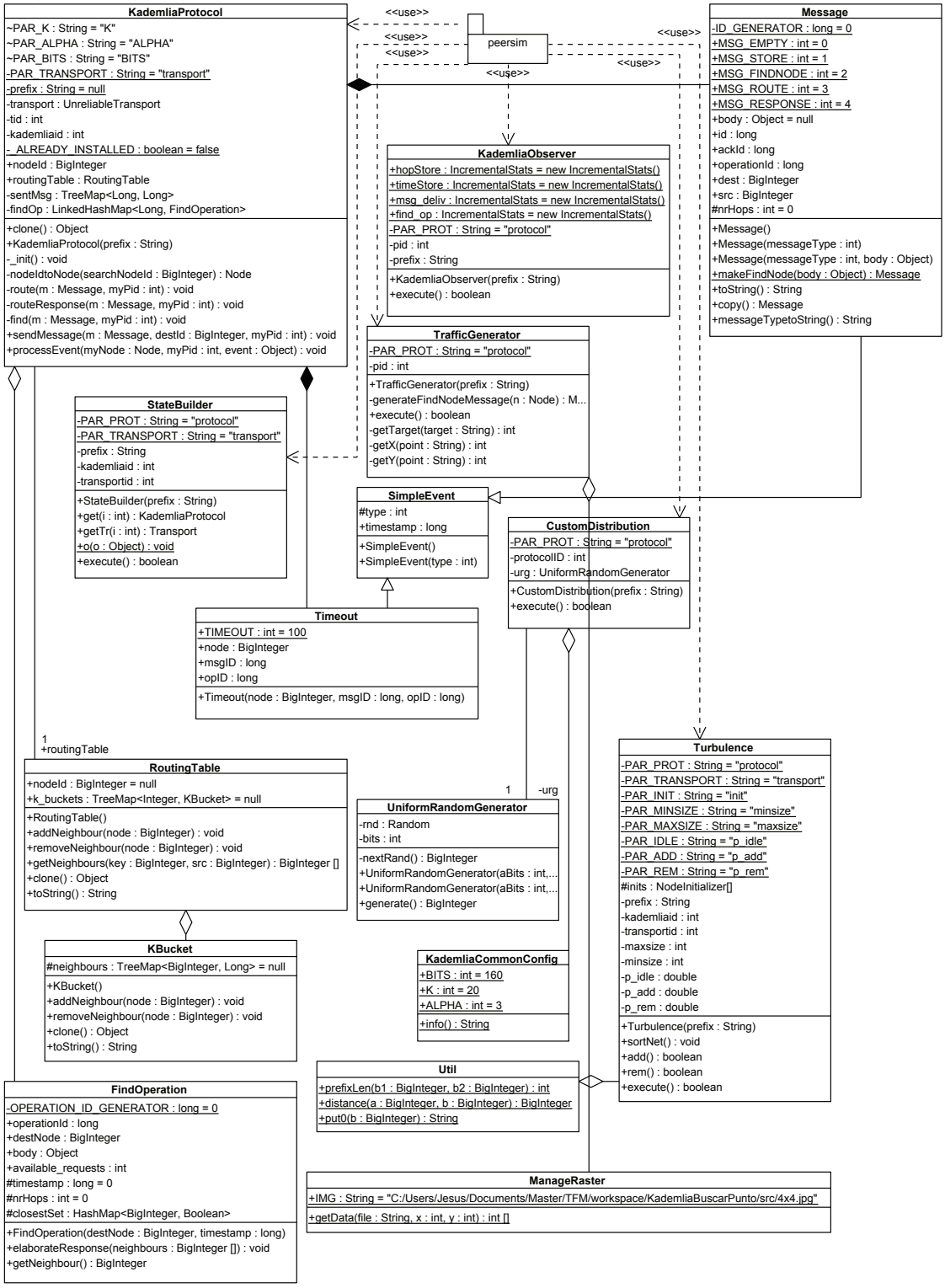

Fig. 25. Diagrama de clases del protocolo *Kademlia* en *PeerSim*.

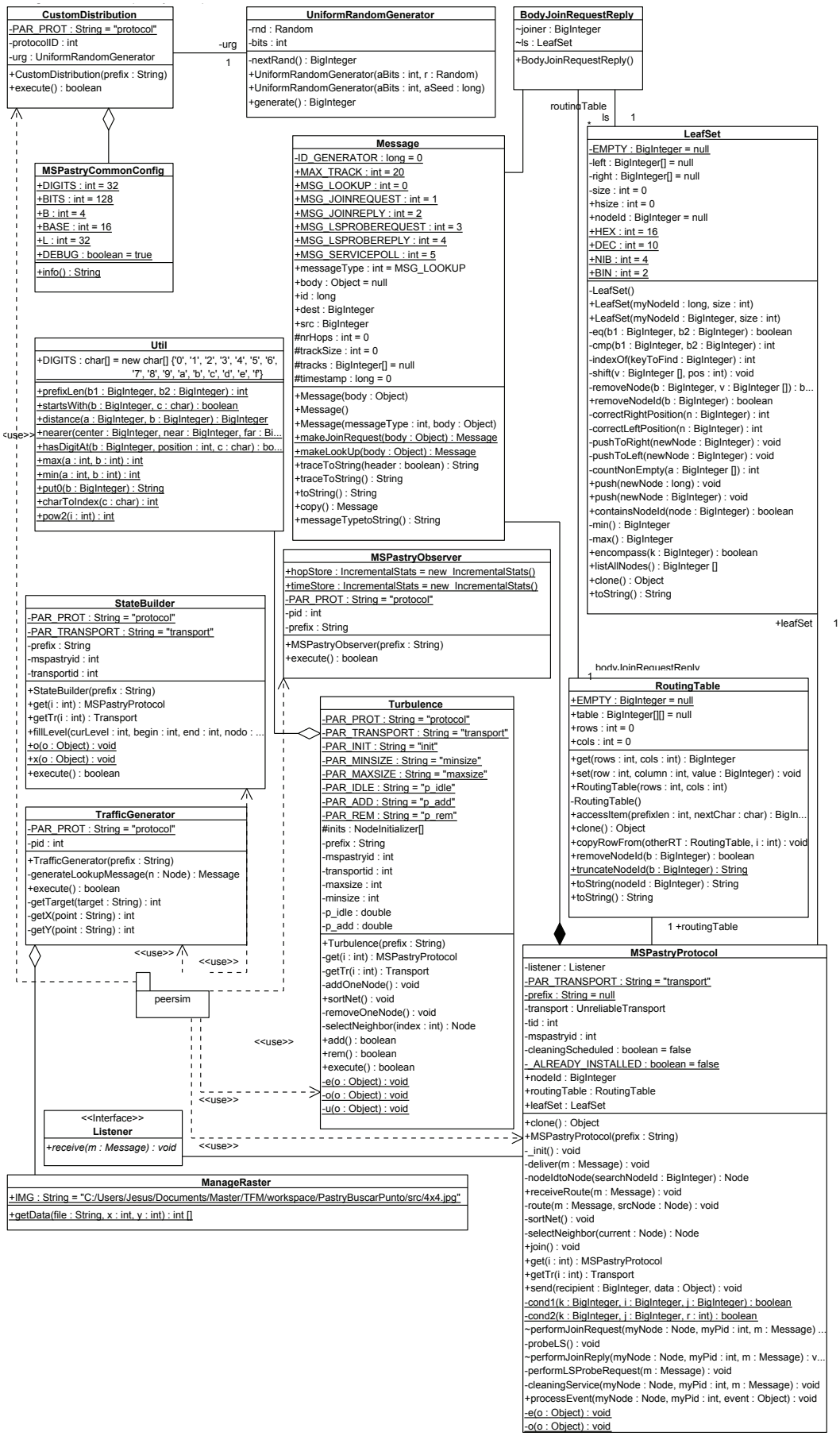

Fig. 26. Diagrama de clases del protocolo *Pastry* en *PeerSim*.

espacial dentro de ellos ha sido necesario linealizar los datos espaciales.

A. Descripción de los datos espaciales

En el desarrollo del trabajo hemos tratado con datos espaciales de dos tipos, ráster y vectorial. La información ráster con la que hemos trabajado se ha basado en una imagen ráster como la que aparece en la Figura 27, que recoge una vista aérea de *Almería* y *Granada*. Esta imagen que fue tomada desde satélite, se le dieron diferentes resoluciones (1024, 2048, 4096 y 8192).

Fig. 27. Imagen ráster de Almería y Granada (resolución $1024 * 1024$).

Para la información vectorial se ha trabajado con una base de datos geográfica construida sobre PostGres[6] a la que le fue instalado PostGis[7]. Esta base de datos contiene una tabla como la que se ve en la Tabla II, que almacena polígonos que describen las provincias de Andalucía y cada punto que forma parte de los polígonos se representa como latitud y longitud.

B. Linealización de la información espacial en nuestros experimentos

Para el linealizado de la información espacial tanto en modo ráster como en modo vectorial, se ha hecho uso de un método de linealización similar al *column-major order* visto en la Sección V-A2. Por tanto, vamos a observar como se ha linealizado la información almacenada en forma matricial para pasar a una estructura de datos uni-dimenaional.

Para ello, con respecto a la información ráster, como la que observar en la Figura 17, vamos a asignar una columna de pixeles de la imagen ráster, de tamaño $n * n$, a cada *par* de la red P2P. Esto significa que para una red de 1024 *pares* la imagen ráster debe de ser de $1024 * 1024$, tal y como se puede observar en la Figura 28 para $n = 1024$.

Así, para el par con identificador 1 le vamos a asignar los píxeles de las posiciones $(1, 1), (1, 2), (1, 3)...(1, n)$ y para el par de la posición n le vamos a asignar los píxeles de las posiciones $(n, 1), (n, 2)...(n, n)$. De esta forma ya hemos

[6]http://www.postgresql.org.es/

[7]http://postgis.net/

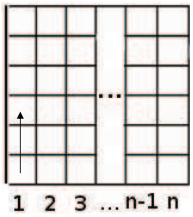

Fig. 28. Imagen ráster de tamaño $n * n$.

conseguido asignar a cada *par* de la red P2P cierto rango de información perteneciente a la imagen ráster para posteriormente poder tratarla y realizar búsquedas puntuales sobre ella. Las consultas en rango y del vecino más cercano en este trabajo no se van a tratar y serán propuestas como trabajo para el futuro.

En el tratamiento de la información vectorial hemos hecho algo similar a como lo hicimos para la información ráster y que se puede ver en la Figura 28. Se ha divido el espacio en vectorial en n columnas, donde n se corresponde al número de *pares* que tiene la red P2P. El tamaño de cada columna en la coordenada X es el espacio total en la coordenada X divido por el número de pares n. Por lo que cuando tengamos que saber qué *par* contiene cierta información geográfica, a través de un sencillo cálculo podremos saber qué *par* lo contiene para enviarle la consulta.

C. Modificaciones realizadas sobre los protocolos

Para poder trabajar con información espacial en estos protocolos aplicándole el método de linealización anteriormente descrito, hemos tenido que realizar unas ligeras modificaciones sobre ellos.

Con respecto al manejo con información ráster, hemos tenido que añadirle a cada protocolo una clase llamada `ManageRaster` que se muestra en la Figura 29. Es una clase que contiene un único método estático al que se le pasa el nombre del fichero que contiene la imagen en formato `.jpg` y la posición del píxel que vamos a consultar. Dentro del método estático llamado `getData` generamos un objeto que contiene la imagen y a partir de él obtenemos el color RGB del píxel que es devuelto.

Otro elemento de la simulación que se ha tenido que modificar ha sido la clase `TrafficGenerator`. En el estado actual para el manejo de información ráster de dicha clase, se obtiene un par aleatorio que va a ser el encargado de partir como origen en el proceso de consulta, luego se obtendrá un par destino que obtendrá del punto que se quiere consultar. Cuando ya se han conocido los pares que intervienen en la búsqueda se lanza la simulación. Cuando se ha localizado el par que tiene el píxel que estamos buscando, se consulta dicho píxel y obtenemos el color que contiene dentro de la imagen. Esta clase se puede observar en la Figura 30.

Para trabajar con la información vectorial, también hemos tenido que modificar el estado de los protocolos. Se ha creado la clase llamada `ManageVector` que gestiona la información vectorial y se ha añadido en todos los protocolos con los que hemos trabajado. Esta clase concretamente, contiene un método estático al cual dado un punto geográfico (localizado

TABLE II. TABLA DE PROVINCIAS DE ANDALUCÍA.

Nombre	Geometría
Almería	'POLYGON(36.7554290 -3.067932, ..., 36.709164 -3.023987)'
Cadiz	'POLYGON(36.787504 -6.30310, ..., 36.542743 -5.493164)'
Cordoba	'POLYGON(38.655488 -2.537842, ..., 38.865375 -5.108643)'
Granada	'POLYGON(36.709164 -3.023987, ..., 38.084851 -2.548828)'
Huelva	'POLYGON(38.522384 -7.212524, ..., 37.987504 -6.300654)'
Jaen	'POLYGON(38.084851 -2.548828, ..., 38.655488 -2.537842)'
Málaga	'POLYGON(36.732281 -3.7902283, ..., 36.304059 -5.039978)'
Sevilla	'POLYGON(37.987504 -6.300659, ..., 37.236889 -4.573059)'

```
public class ManageRaster {
      /**
      * @param file Imagen ráster para devolver el color del pixel
      * @param x Posición de la coordenada X de la imagen
      * @param y Posición de la coordenada Y de la imagen
      * @return int[] Color RGB del pixel
      */
      public static int[] getData(String file, int x, int y) {
            // Objeto que obtiene la imagen
            BufferedImage img = null;
            try{
                  img = ImageIO.read(new File("C:/.../" + file));
            }catch(IOException e){
                  e.printStackTrace();
            }

            // Obtención del color del pixel
            int argb = img.getRGB(x, y);

            // Paso del color del pixel
            int rgb[] = new int[] {
                  (argb >> 16) & 0xff, //red
                  (argb >>  8) & 0xff, //green
                  (argb      ) & 0xff  //blue
            };
            return rgb;
      }
}
```

Fig. 29. Clase `ManageRaster` en lenguaje *Java* para manejo de información ráster.

en Andalucía) con latitud y longitud es capaz de conectarse a una base de datos geográfica, lanzar una consulta y obtener la provincia de Andalucía que contiene dicho punto. Esta clase se puede observar en la Figura 31.

La clase `TrafficGenerator` también la hemos modificado para el manejo de información vectorial al igual que para trabajar con la información ráster, estando visible en la Figura 32. El funcionamiento de esta clase es la siguiente, se obtiene un punto aleatorio que va a ser el encargado de partir como origen en el proceso de consulta, luego se obtendrá un par destino a partir de la latitud del punto que se desea consultar. El par destino se calcula en el método `getPeer` donde dados el número de pares y la distancia geográfica que controla cada par calcula que par contiene la información para el punto espacial facilitado. Cuando ya se han conocido los pares que intervienen en la búsqueda se lanza la simulación. Cuando se ha localizado el par que tiene la información vectorial, se consulta la base de datos geográfica por medio de una consulta como la siguiente `select name from provincias where ST_Contains(the_geom, GeomFromText('POINT(37.1,-4)',3395))`. Esta consulta busca el polígono que contiene dicho punto obteniendo de esta forma la provincia correspondiente.

```
public class TrafficGenerator implements Control {
      ...
      /*
       * Método que lanza la simulación
       */
      public boolean execute() {
            // Consultar el color del pixel
            boolean seeColor = false;
            // Obteniendo el tamaño de la red
            int size = Network.size();
            Node sender, target = null;
            // Imagen que vamos a consultar
            String file = "4x4.jpg";
            // Punto que vamos a consultar
            String point = "(2,2)";

            // Par emisor de consulta
            sender = Network.get(CommonState.r.nextInt(size));

            try{
                  // Par receptor de consulta, obtenido a
                  // partir de las coordenadas del punto.
                  target = Network.get(getTarget(point));

                  // Generar el mensaje
                  LookUpMessage message = new LookUpMessage(...);

                  // Lanzar simulación
                  EDSimulator.add(0, message, sender, pid);
            }catch(Exception e){
            }

            // Obtención del punto en la imagen
            if(target != null && seeColor){
                  int[] image = ManageRaster.getData(file,

      /*X*/, /*Y*/);
            }
            return false;
      }

      /*
      * Métodos getTarget, getX, getY
      */
}
```

Fig. 30. Clase `TrafficGenerator` en lenguaje *Java* para manejo de información ráster.

```
public class ManageVector {

      public static String getData(String point) {
            // Provincia que contiene el punto
            String provincia = "";
            // Consulta a lanzar sobre la BD Geográfica
            String consulta = "select name …)";
            // Conexión con la BD
            String cc = "jdbc:postgresql:…";

            ...

            Connection conexion = DriverManager.
                                    getConnection(cc);
            Statement comando = conexion.
                                    createStatement();
            // Obtenemos los resultados
            ResultSet resultado = comando.
                              executeQuery(consulta);
            resultado.next();
            // Obtenemos la provincia
            provincia = resultado.getString(1);

            ...

            return provincia;
      }
}
```

Fig. 31. Clase `ManageVector` en lenguaje *Java* para manejo de información vectorial.

```
public class TrafficGenerator implements Control {
      ...
      /*
      * Método que lanza la simulación
      */
      public boolean execute() {
            // Consultar el color del pixel
            boolean seePeer = true;
            // Calcular el tamaño de la red
            int size = Network.size();
            Node sender, target = null;

            // Par emisor de consulta
            sender = Network.get(CommonState.r.nextInt(size));

            try{
                  // Par receptor de consulta   LATITUD  LONGITUD
                  target = Network.get(getPeer(…);
                  // Generar el mensaje
                  LookUpMessage message = new LookUpMessage(...);
                  // Lanzar simulación
                  EDSimulator.add(0, message, sender, pid);
            }catch(Exception e){
            }

            // Obtención del punto
            if(target != null && seePeer){
                        int par = getPeer(1024, "37.1", "-4");
                        ManageVector.action("(37.1 -4)");
            }
            return false;
      }
      /*
      * Método getPeer
      */
}
```

Fig. 32. Clase `TrafficGenerator` en lenguaje *Java*, para manejo de información vectorial.

D. Métricas de la simulación

Como ya hemos mencionado en secciones previas se ha trabajado con los protocolos *Pastry*, *Chord* y *Kademlia*. Para las simulaciones de los mismos hemos utilizado librerías sobre *PeerSim* donde ya venían implementados estos protocolos siendo necesario modificar algunas clases. Para llevar a cabo comparativas y poder ver como se comportaba cada protocolo debemos previamente observar qué medida era la más apropiada para cada uno. Todos tenían en cuenta el número de saltos que da un mensaje hasta llegar a su destinatario, y hemos adaptado *Chord* con el objetivo de medir también el tiempo de simulación.

- Para *Chord* la simulación recoge el *número de fallos* que se producen en la red al no localizar el destinatario al que va dirigido el mensaje, por agregación o eliminación de nodos y no estar actualizadas las listas de sucesores. También mide la *estabilización* encargada de medir la tolerancia a fallos de la red [Xu and Srimani, 2005]. Después se calcula la *media* del número de mensajes recibidos en cada par que forma la red o lo que es lo mismo el número de saltos que da un mensaje hasta llegar a su destino, el *máximo* de mensajes que ha recibido un par en la red y el *mínimo* de mensajes que ha recibido un par en la red (salida por consola de una simulación: `[Mean: 6.0 Max Value: 12 Min Value: 0 Observations: 900 Stabilizations: 0 Failures: 0]`). Aunque este protocolo no mida el tiempo de simulación en la modificación de la clase `TrafficGenerator` se han introducido instrucciones que permiten medirlo.
- El protocolo *Pastry* en el simulador mide el *tiempo de simulación* del experimento, el *número de pares* que están trabajando en la red para esa simulación, la *media de saltos* que se han producido en la red y el *tiempo medio* de transmitir un mensaje (salida por consola de una simulación: `[time=290000]:[with N=1015 current node UP][2,685824 average hops][1880 msec average trasimission time]`).
- Para el protocolo *Kademlia* hemos medido el *tiempo de simulación* hasta localizar el mensaje, el *número de nodos* que forman la red, el *porcentaje de mensajes* entregados de forma correcta a su destinatario, el *mínimo de saltos* que se ha producido para un mensaje de la red, la *media de saltos* que se ha producido para los mensajes en la red y el *máximo de saltos* que se ha producido para un mensaje de la red (salida por consola de una simulación: `[time=11220]:[N=32767 current node UP][D=0,00000 msg deliv][Infinity min h][NaN average h][-Infinity max h]`).

Hemos realizado experimentos con dos tipos de información espacial, ráster y vectorial. Así que tanto para la información ráster como para la vectorial se han realizado dos tipos de

mediciones, una de ellas basada en el número de saltos que realiza un mensaje en la red hasta localizar el par que contiene la información y la otra en el tiempo que tarda el protocolo en localizar el par que contiene el dato que estamos buscando.

La Figura 33 muestra el número de saltos que ha dado un mensaje en cada protocolo para localizar un punto en una imagen ráster cuya resolución ha variado para cada lanzamiento del experimento. Las resoluciones de la imagen son de $n * n$, donde n además será el número de pares que forman la red, tomando n los valores $1024, 2048, 4096, 8192$. Por tanto, en la Figura 33 se puede observar el protocolo que menos saltos realiza y que mejores resultados ha obtenido ha sido *Pastry* con menos de tres saltos aproximadamente para un número de pares de 8192 para una imagen de $8192 * 8192$. *Chord* por otro lado se distancia un poco de *Pastry*, moviendose en un rango de entre tres y cinco saltos para localizar puntos en imágenes que van desde $1024 * 1024$ hasta $8192 * 8192$. El protocolo en el que más saltos realizan los mensajes es *Kademlia* donde se llegan a cerca de treinta mensajes para una imagen de $8192 * 8192$. Que el número de saltos sea menor para *Chord* y *Pastry* no es coincidencia, en ambos el espacio de claves esta ordenado de manera circular. Por otro lado, *Kademlia* utiliza un árbol binario lleno para organizar las claves de búsqueda lo que hace que un mensaje tenga que dar más saltos hasta llegar al destino. Luego *Pastry* le gana la batalla a *Chord* en cuanto al número de saltos que da un mensaje en la red P2P antes de llegar al destino, pues la *Tabla de encaminamiento* de *Pastry* consigue hacer más corto el camino a seguir por el mensaje frente a la *Tabla de apuntadores* que usa *Chord* (ver Sección IV donde se explica el re-direccionamiento de los mensajes en cada protocolo).

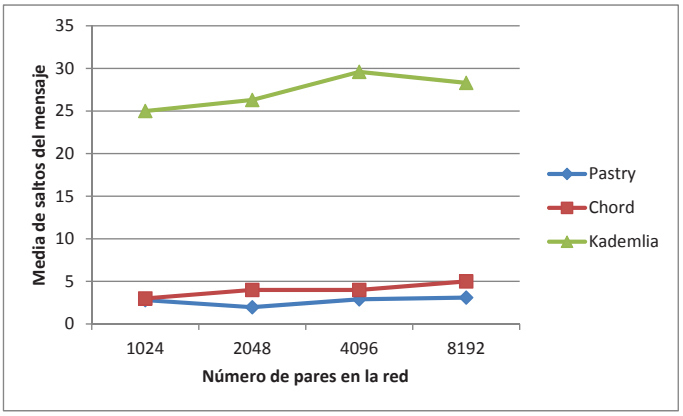

Fig. 33. Saltos para consultas puntuales en información ráster.

En la Figura 34 mostramos los tiempos de ejecución que hemos obtenido al simular los protocolos *Pastry*, *Kademlia* y *Chord* para la búsqueda de información en una imagen ráster. Hemos trabajado con imágenes como la que se observa en la Figura 27 con resoluciones comprendidas entre $1024 * 1024, 2048 * 2048, 4096 * 4096$ y $8192 * 8192$.

En dicha gráfica no podemos observar claramente cuál de los protocoles es más rápido para localizar un pixel en la imagen. Se debe a que el tiempo de acceso a la imagen es

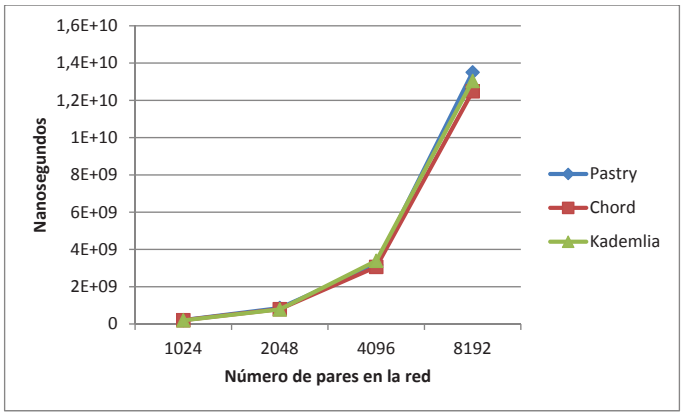

Fig. 34. Tiempo total para consultar un punto en una imagen ráster en cada protocolo.

tan elevado que deja invisible el tiempo de búsqueda en la red. Para poder ver los tiempos de búsqueda de información ráster en la red debemos irnos a la Figura 35. Donde ya podemos ver claramente como *Pastry* vuelve a ser más rápido que *Chord* y *Kademlia*, al devolver los resultados antes para los casos en los que se trabajó con imágenes con resoluciones de $1024 * 1024, 2048 * 2048$ y $8192 * 8192$. *Chord* a su vez es más rápido que *Kademlia* al devolver los resultados antes para imágenes con resoluciones de $2048 * 2048, 4096 * 4096$ y $8192 * 8192$. De este modo, se vuelve a cumplir de cierta manera lo observado en la Figura 33 pues *Pastry* tiene un mecanismo de encaminamiento más eficiente.

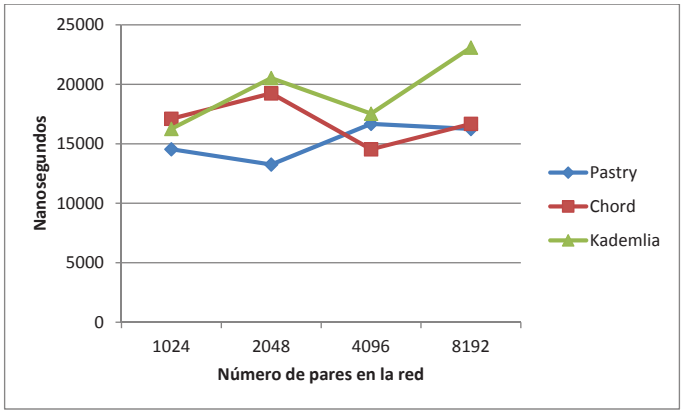

Fig. 35. Tiempo de búsqueda del par que contiene la información ráster en cada protocolo.

La Figura 36 muestra el número de saltos que ha dado un mensaje en cada protocolo para localizar un punto dentro de la base de datos geográfica con datos vectoriales descrita en la Sección IX-A. La tendencia de los resultados obtenidos es muy similar a la obtenida en la localización de un punto en una imagen ráster según la Figura 33, ya que la búsqueda de un par en la red, en nuestro caso, es la misma independientemente de la información de la que se trate dentro de ella. Vuelve a ser *Pastry* el protocolo que menos saltos realiza y que mejores

resultados ha obtenido con menos de cinco saltos para un número de pares de 32768. *Chord* por otro lado, continua estando por encima de *Pastry*. El protocolo en el que más saltos realizan los mensajes vuelve a ser *Kademlia* que se ha movido entre veinticuatro y treinta y un mensajes antes de localizar el par. La explicación a estas diferencias de saltos en los mensajes que recorren la red para la búsqueda en información vectorial son las mismas que comentamos anteriormente para la búsqueda de un punto en una imagen ráster.

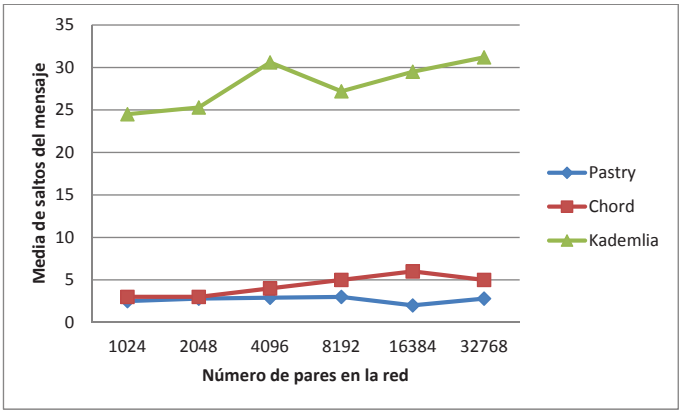

Fig. 36. Saltos para consultas puntuales en información vectorial.

En la Figura 37 mostramos los tiempos que hemos obtenido al simular los protocolos *Pastry*, *Kademlia* y *Chord* al igual que en la Figura 34. El resultado de dichos tiempos es practicamente el mismo que para cualquiera de los tres protocolos, es decir, el tiempo de simular la búsqueda de información en el sistema P2P y la devolución de esta información es muy similar para todos. Esto se debe a que el tiempo de buscar información en la red es irrelevante con respecto al tiempo en que se tarda en acceder a la información de la base de datos geográfica, por lo que no se puede apreciar cual de los tres es más rápido ya que la parte más costosa es el acceso a la información en la base de datos. También podemos comparar estos tiempos con los accesos a un punto en una imagen ráster, para comprobar que acceder a información vectorial almacenada en una base de datos geográfica tiene un tiempo de menor que el acceso a puntos de una imagen ráster.

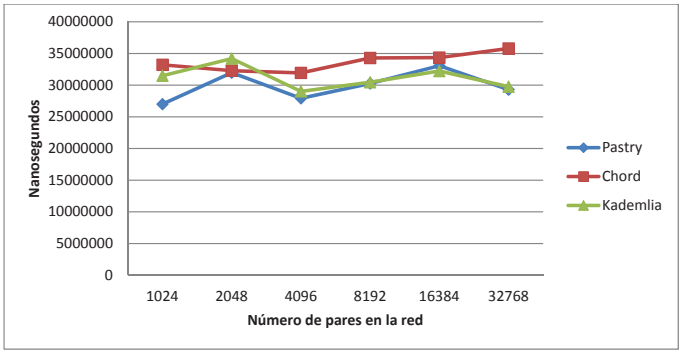

Fig. 37. Tiempo total para consultar un punto en una imagen vectorial en cada protocolo.

Con el objetivo de observar con suficiente precisión que protocolo tarda menos tiempo en acceder a un punto espacial en información vectorial, esta la Figura 38. En esta Figura se observa con menos claridad que *Pastry* es más rápido que el resto de protocolos, pero para los casos de acceso a información para una red de 1024, 2048 y 32768 pares es capaz de obtener mejores resultados. Esta mejora de los resultado se debe a como *Pastry* re-direcciona los mensajes en la red.

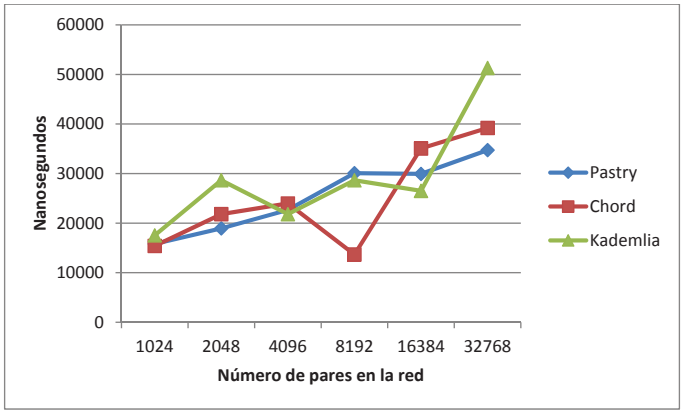

Fig. 38. Tiempo de búsqueda del par que contiene la información vectorial en cada protocolo.

Concluyendo, se ha podido comprobar según las simulaciones que el protocolo *Pastry* es más eficiente frente a *Chord* y *Kademlia*. Todo ello debido al modo en el cuál *Pastry* realiza el re-direccionamiento de la información dentro de la red P2P. Observamos además que los accesos, en cuanto a tiempo, sobre la información tanto ráster como vectorial es bastante elevado. Aunque según los experimentos, consultar un punto situado en una imagen ráster es más costoso que hacerlo sobre una base de datos geográfica que contiene información ráster.

X. Conclusiones y trabajos futuros

Con este trabajo pretendemos hacer uso de las ventajas de los sistemas P2P, para aplicarlos a información espacial. Para ello, hay que realizar un tratamiento previo de la información espacial para poder integrarla dentro de este tipo de redes, pasando dicha información de un espacio de 2D a 1D utilizando algunos de los métodos que hemos descrito. Además, hemos podido comprobar la diferencia a la hora de trabajar con diferentes protocolos para la red P2P en la que hemos integrado la información espacial. Para llevar a cabo las comprobaciones, previamente se ha realizado un estudio de las herramientas que permiten simular los sistemas P2P, de las cuales hemos seleccionado *PeerSim* por facilitar la reutilización de protocolos ya desarrollados por medio de librerías. También hemos podido observar gracias a los resultados de los experimentos con consultas puntuales, qué protocolos son más eficientes, en cuanto al número de saltos que se producen en la red, destacando a *Pastry* y *Chord*. También los tiempos en los cuales se ejecuta cada experimento realizado sobre el simulador *PeerSim* y como es de costoso el acceso a información vectorial gestionada por una base de

datos geográfica con respecto a la búsqueda de pares en la red P2P.

Como hemos mencionado ya en la Sección VII-A el simulador *PeerSim* tiene ciertas ventajas, pero no hemos mencionado algunos de sus inconvenientes. Destacamos que sólo puede simular una red de 10^6 pares cuando usa una simulación basada en ciclos, modo de simulación que además no tiene en cuenta la capa de transporte. La simulaciones basadas en eventos son más realistas que las simulaciones basadas en ciclos pero no son muy usadas ya que son más complejas de implementar. A lo que hay que añadir que *PeerSim* no es un simulador que esté muy documentado.

Como futuros trabajos, poder realizar consultas de rangos sobre los datos y no solo buscar en la red P2P elementos puntuales como ha sido un pixel en una imagen ráster o un punto geográfico en un espacio vectorial sino también la consultas relacionadas con el vecino más cercano. Además poder modificar algún protocolo como *Chord* en el cual se distribuya la información de forma coherente de tal manera que elementos que sean cercanos en el espacio sean almacenados en pares cercanos dentro de la red, como se ha desarrollado en [Shu et al., 2005] haciendo uso de *Z-order*. Otro tema interesante sería trabajar con grandes cantidades de información espacial para ver como se comportan los protocolos con redes de más pares. También como tarea futura sería interesante trabajar con información espacial en el contexto de *cloud computing*, con el objetivo de poder llevar a cabo, dentro de este tipo de entornos, consultas de rango y puntuales para datos espaciales al igual que consiguieron en [Wang et al., 2010].

REFERENCES

[Barbosa et al., 2007] Barbosa, J., Hahn, R., Bonatto, D., Cecin, F., and Geyer, C. (2007). Evaluation of a large-scale ubiquitous system model through peer-to-peer protocol simulation. In *Distributed Simulation and Real-Time Applications, 2007. DS-RT 2007. 11th IEEE International Symposium*, pages 175–181. IEEE.

[Baumgart et al., 2007] Baumgart, I., Heep, B., and Krause, S. (2007). Oversim: A flexible overlay network simulation framework. In *IEEE Global Internet Symposium, 2007*, pages 79–84. IEEE.

[Berman et al., 2003] Berman, F., Fox, G., and Hey, A. J. (2003). *Grid computing: making the global infrastructure a reality*, volume 2. Wiley. com.

[Bharambe et al., 2004] Bharambe, A. R., Agrawal, M., and Seshan, S. (2004). Mercury: supporting scalable multi-attribute range queries. *ACM SIGCOMM Computer Communication Review*, 34(4):353–366.

[Bischofs and Hasselbring, 2009] Bischofs, L. and Hasselbring, W. (2009). Analyzing and implementing peer-to-peer systems with the peerse experiment environment. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 311–315. IEEE.

[Cai et al., 2004] Cai, M., Frank, M., Chen, J., and Szekely, P. (2004). Maan: A multi-attribute addressable network for grid information services. *Journal of Grid Computing*, 2(1):3–14.

[CHÁVEZ,] CHÁVEZ, A. G. M. Evaluación de la confiabilidad de índices p2p en presencia de alta transitoriedad.

[Chien et al., 2003] Chien, A., Calder, B., Elbert, S., and Bhatia, K. (2003). Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5):597–610.

[Chtepen et al., 2009] Chtepen, M., Claeys, F. H., Dhoedt, B., De Turck, F., Demeester, P., and Vanrolleghem, P. A. (2009). Adaptive task checkpointing and replication: Toward efficient fault-tolerant grids. *Parallel and Distributed Systems, IEEE Transactions on*, 20(2):180–190.

[Crainiceanu et al., 2004] Crainiceanu, A., Linga, P., Gehrke, J., and Shanmugasundaram, J. (2004). Querying peer-to-peer networks using p-trees. In *Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004*, pages 25–30. ACM.

[Domingues et al., 2009] Domingues, P., Araujo, F., and Silva, L. (2009). Evaluating the performance and intrusiveness of virtual machines for desktop grid computing. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE.

[Eastlake and Jones, 2001] Eastlake, D. and Jones, P. (2001). Us secure hash algorithm 1 (sha1).

[Foster and Iamnitchi, 2003] Foster, I. and Iamnitchi, A. (2003). On death, taxes, and the convergence of peer-to-peer and grid computing. In *Peer-to-Peer Systems II*, pages 118–128. Springer.

[Ganesan et al., 2004] Ganesan, P., Yang, B., and Garcia-Molina, H. (2004). One torus to rule them all: multi-dimensional queries in p2p systems. In *Proceedings of the 7th International Workshop on the Web and Databases: colocated with ACM SIGMOD/PODS 2004*, pages 19–24. ACM.

[García et al., 2005] García, P., Pairot, C., Mondéjar, R., Pujol, J., Tejedor, H., and Rallo, R. (2005). Planetsim: A new overlay network simulation framework. In *Software engineering and middleware*, pages 123–136. Springer.

[Gil et al., 2009] Gil, J.-M., Song, U.-S., and Yu, H.-C. (2009). Performance evaluation of scheduling mechanism with checkpoint sharing and task duplication in p2p-based pc grid computing. In *Advances in Grid and Pervasive Computing*, pages 459–470. Springer.

[González et al., 2009] González, D. L., de Vega, F. F., Trujillo, L., Olague, G., Araujo, L., Castillo, P., Merelo, J. J., and Sharman, K. (2009). Increasing gp computing power for free via desktop grid computing and virtualization. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 419–423. IEEE.

[Gummadi et al., 2003] Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., and Stoica, I. (2003). The impact of dht routing geometry on resilience and proximity. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 381–394. ACM.

[Gupta et al., 2003] Gupta, I., Birman, K., Linga, P., Demers, A., and Van Renesse, R. (2003). Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Peer-to-Peer Systems II*, pages 160–169. Springer.

[Guttman, 1984] Guttman, A. (1984). *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM.

[Heien et al., 2009] Heien, E. M., Anderson, D. P., and Hagihara, K. (2009). Computing low latency batches with unreliable workers in volunteer computing environments. *Journal of Grid Computing*, 7(4):501–518.

[Ho et al., 2008] Ho, R. S., Wang, C.-L., and Lau, F. (2008). Lightweight process migration and memory prefetching in openmosix. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12. IEEE.

[Jagadish et al., 2006] Jagadish, H., Ooi, B. C., Vu, Q. H., Zhang, R., and Zhou, A. (2006). Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 34–34. IEEE.

[Jagadish et al., 2005] Jagadish, H. V., Ooi, B. C., and Vu, Q. H. (2005). Baton: A balanced tree structure for peer-to-peer networks. In *Proceedings of the 31st international conference on Very large data bases*, pages 661–672. VLDB Endowment.

[Kaashoek and Karger, 2003] Kaashoek, M. F. and Karger, D. R. (2003). Koorde: A simple degree-optimal distributed hash table. In *Peer-to-Peer Systems II*, pages 98–107. Springer.

[Kacsuk et al., 2007] Kacsuk, P., Podhorszki, N., and Kiss, T. (2007). Scalable desktop grid system. In *High Performance Computing for Computational Science-VECPAR 2006*, pages 27–38. Springer.

[Kamel and Faloutsos, 1993] Kamel, I. and Faloutsos, C. (1993). Hilbert r-tree: An improved r-tree using fractals.

[Kantere et al., 2009] Kantere, V., Skiadopoulos, S., and Sellis, T. (2009). Storing and indexing spatial data in p2p systems. *Knowledge and Data Engineering, IEEE Transactions on*, 21(2):287–300.

[Karger et al., 1997] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. (1997). Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM.

[Lavastida-López and Almeida-Cruz, 2009] Lavastida-López, Z. and Almeida-Cruz, Y. (2009). Propuesta de un modelo para el intercambio automático de información en redes p2p.

[Lee et al., 2005] Lee, J., Lee, H., Kang, S., Choe, S., and Song, J. (2005). Ciss: An efficient object clustering framework for dht-based peer-to-peer applications. In *Databases, Information Systems, and Peer-to-Peer Computing*, pages 215–229. Springer.

[Lua et al., 2005] Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7(1-4):72–93.

[Malkhi et al., 2002] Malkhi, D., Naor, M., and Ratajczak, D. (2002). Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192. ACM.

[Marosi et al., 2009] Marosi, A. C., Balaton, Z., and Kacsuk, P. (2009). Genwrapper: a generic wrapper for running legacy applications on desktop grids. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–6. IEEE.

[Maymounkov and Mazieres, 2002] Maymounkov, P. and Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer.

[Mondal et al., 2005] Mondal, A., Lifu, Y., and Kitsuregawa, M. (2005). P2pr-tree: An r-tree-based spatial index for peer-to-peer environments. In *Current Trends in Database Technology-EDBT 2004 Workshops*, pages 516–525. Springer.

[Orenstein and Merrett, 1984] Orenstein, J. A. and Merrett, T. H. (1984). A class of data structures for associative searching. In *Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 181–190. ACM.

[Pérez-Miguel et al., 2009] Pérez-Miguel, C., Miguel-Alonso, J., and Mendiburu, A. (2009). Informe sobre sistemas de computación en redes p2p. Technical report, EHU.

[Plaxton et al., 1999] Plaxton, C. G., Rajaraman, R., and Richa, A. W. (1999). Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32(3):241–280.

[Pourebrahimi et al., 2005] Pourebrahimi, B., Bertels, K., and Vassiliadis, S. (2005). A survey of peer-to-peer networks. In *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing, ProRisc*, volume 2005. Citeseer.

[Ratnasamy et al., 2001] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. (2001). *A scalable content-addressable network*, volume 31. ACM.

[Rigaux et al., 2001] Rigaux, P., Scholl, M., and Voisard, A. (2001). *Spatial databases: with application to GIS*. Morgan Kaufmann.

[Rowstron and Druschel, 2001] Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer.

[Sahin et al., 2005] Sahin, O. D., Antony, S., Agrawal, D., and El Abbadi, A. (2005). Probe: Multi-dimensional range queries in p2p networks. In *Web Information Systems Engineering–WISE 2005*, pages 332–346. Springer.

[Samet, 2006] Samet, H. (2006). *Foundations of multidimensional and metric data structures*. Morgan Kaufmann.

[Santhanam et al., 2005] Santhanam, S., Elango, P., Arpaci-Dusseau, A., and Livny, M. (2005). Deploying virtual machines as sandboxes for the grid. In *Second Workshop on Real, Large Distributed Systems (WORLDS 2005)*.

[Schmidt and Parashar, 2003] Schmidt, C. and Parashar, M. (2003). Flexible information discovery in decentralized distributed systems. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pages 226–235. IEEE.

[Shu et al., 2005] Shu, Y., Ooi, B. C., Tan, K.-L., and Zhou, A. (2005). Supporting multi-dimensional range queries in peer-to-peer systems. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 173–180. IEEE.

[Stoica et al., 2001] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Computer Communication Review*, volume 31, pages 149–160. ACM.

[Tanin et al., 2007] Tanin, E., Harwood, A., and Samet, H. (2007). Using a distributed quadtree index in peer-to-peer networks. *The VLDB Journal*, 16(2):165–178.

[Wang et al., 2010] Wang, J., Wu, S., Gao, H., Li, J., and Ooi, B. C. (2010). Indexing multi-dimensional data in a cloud system. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 591–602. ACM.

[Wei and Sezaki, 2006] Wei, X. and Sezaki, K. (2006). Dhr-trees: A distributed multidimensional indexing structure for p2p systems. In *Parallel and Distributed Computing, 2006. ISPDC'06. The Fifth International Symposium on*, pages 281–290. IEEE.

[Xie and Bi, 2008] Xie, Z. and Bi, J. (2008). Peerstrategy: A local strategy for peers to evaluate their neighbors. In *Grid and Cooperative Computing, 2008. GCC'08. Seventh International Conference on*, pages 386–391. IEEE.

[Xu and Srimani, 2005] Xu, Z. and Srimani, P. K. (2005). Self-stabilizing publish/subscribe protocol for p2p networks. In *Distributed Computing–IWDC 2005*, pages 129–140. Springer.

[Yianilos and Sobti, 2001] Yianilos, P. N. and Sobti, S. (2001). The evolving field of distributed storage. *Internet Computing, IEEE*, 5(5):35–39.

[Zhang et al., 2011] Zhang, C., Xiao, W., Tang, D., and Tang, J. (2011). P2p-based multidimensional indexing methods: A survey. *Journal of Systems and Software*, 84(12):2348–2362.

[Zhang et al., 2009] Zhang, H., Jin, H., and Zhang, Q. (2009). Scheduling strategy of p2p based high performance computing platform base on session time prediction. In *Advances in Grid and Pervasive Computing*, pages 364–375. Springer.

[Zhao et al., 2001] Zhao, B. Y., Kubiatowicz, J., Joseph, A. D., et al. (2001). Tapestry: An infrastructure for fault-tolerant wide-area location and routing.

[Zhao et al., 2005] Zhao, S., Lo, V., and Dickey, C. (2005). Result verification and trust-based scheduling in peer-to-peer grids. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 31–38. IEEE.

[Zurita, 2011] Zurita, L. (2011). *La Gestión del conocimiento territorial*. Ra-Ma.

UNIVERSIDAD DE ALMERÍA

Resumen

La gestión de datos espaciales es útil hoy en día en muchos campos de aplicación, tales como la geociencia, CAD, la robótica o la protección del medio ambiente por mencionar algunos. Tratar con este tipo de datos (puntos, líneas, polígonos, regiones, etc.) puede llegar a ser muy costoso. Por ello, surgieron los métodos de Indexación Espacial (IE) con el objetivo de mejorar la eficiencia de las consultas en este tipo de datos. Con respecto a los sistemas Peer-to-Peer (P2P), estamos ante una alternativa a los sistemas distribuidos ya que son redes dinámicas que tienen la finalidad de compartir recursos de forma distribuida. Los recursos van desde archivos multimedia, datos, ciclos de procesamiento, etc. Este sistema está formado por *pares* los cuales consiguen dotar a la red de ventajas como: *autonomía*, ya que los pares no siguen regla alguna sobre la manera con la que deben de comportarse en el sistema y como deben de compartir los recursos; *simetría*, pues los pares pueden actuar como clientes, haciendo uso de los recursos que otros pares comparten. Con este trabajo pretendemos hacer uso de las ventajas de los sistemas P2P, para aplicarlos a información espacial. Para ello, como se redacta a lo largo del documento, hay que realizar un tratamiento previo de la información espacial para poder integrarla dentro de este tipo de redes, pasando dicha información de un espacio de 2D a 1D utilizando algunos de los métodos que aparecen a continuación. Además, podremos comprobar la diferencia a la hora de trabajar con diferentes protocolos P2P para la red en la que hemos integrado la información espacial. Para llevar a cabo nuestro estudio, previamente se ha realizado una revisión de las herramientas que permiten simular los sistemas P2P, de las cuales hemos seleccionado *PeerSim* por facilitar la reutilización de protocolos ya desarrollados por medio de librerías y estar desarrollado en Java. En dicho simulador se ha trabajo con protocolos P2P como *Pastry*, *Kademlia* y *Chord* sobre los que hemos simulado el acceso a información espacial de tipo ráster y vectorial. Por citar un ejemplo, en uno de los experimentos realizados entre los protocolos, hemos medido la diferencia que existe entre aumentar el número de pares que gestionan la información vectorial y ráster, para buscar un punto en el espacio con el objetivo de observar el número de saltos que tuvo lugar en la red para localizar la información.