Universidad de Murcia
Department of Information and Communication Engineering

Development of distributed algorithms for data search and content distribution in structured peer-to-peer networks

A thesis submitted for the degree of

PhilosophiæDoctor (PhD)

Presented by:

Jordi Pujol Ahulló

Directed by:

Dr. Pedro García López

Department of Computer Science
Engineering and Maths
Universitat Rovira i Virgili

Dr. Antonio F. Gómez Skarmeta

Department of Information and
Communication Engineering
Universidad de Murcia

2009 November

D. Pedro García López, Profesor Titular de Universidad del Área de Telemática en el Departamento de Ingeniería Informática y Matemáticas, de la Universidad Rovira i Virgili, y D. Antonio F. Gómez Skarmeta, Profesor Titular de Universidad del Área de Telemática en el Departamento de Ingeniería de la Información y las Comunicaciones, de la Universidad de Murcia, AUTORIZAN:

La presentación de la Tesis Doctoral titulada "Desarrollo de Algorithmos Distribuidos para la Búsqueda de Información y Distribución de Contenidos en Redes Estructuradas Peer-to-Peer", realizada por D. Jordi Pujol Ahulló, bajo nuestra inmediata dirección y supervisión, y que presenta para la obtención del grado de Doctor por la Universidad de Murcia.

En Murcia, a 6 de noviembre de 2009

Pedro García López
Antonio F. Gómez Skarmeta

D. Luis Daniel Hernández Molinero, Profesor Titular de Universidad del área de Ingeniería Telemática y Director del Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia, INFORMA:

Que la Tesis Doctoral titulada "Desarrollo de Algorithmos Distribuidos para la Búsqueda de Información y Distribución de Contenidos en Redes Estructuradas Peer-to-Peer", ha sido realizada por D. Jordi Pujol Ahulló, bajo la inmediata dirección y supervisión de D. Pedro García López y D. Antonio F. Gómez Skarmeta, y que el Departamento ha dado su conformidad para que sea presentada ante la Comisión de Doctorado.

En Murcia, a de de 2009

D. Luis Daniel Hernández Molinero

TESIS DOCTORAL:	Desarrollo de Algorithmos Distribuidos para la Búsqueda de Información y Distribución de Contenidos en Redes Estructuradas Peer-to-Peer
AUTOR:	D. Jordi Pujol Ahulló
CO-DIRECTORES:	Dr. Pedro García López Dr. Antonio F. Gómez Skarmeta

El tribunal nombrado para juzgar la Tesis arriba indicada, compuesto por los siguientes doctores:

PRESIDENTE:

VOCALES:

SECRETARIO:

acuerda otorgarle la calificación de:

En Murcia, a de 20 .

El Secretario del Tribunal

A jade stone is useless before it is processed;
a man is good-for-nothing until he is educated.
Chinese proverb

To Barbarakay Cisterna with love

Acknowledgements

I would like to give my sincere thanks to my advisors Prof. Pedro García López and Prof. Antonio F. Gómez Skarmeta for their support. They provided me a total freedom to choose the subject of research, after which they gave me their total support and encouraged me until the conclusion of this dissertation. Their rattling good personality has been a key stone in all the evolution of my research.

I also thank very much to all friends from our Laboratory of Architecture and Telematic Services (ATS) at Universitat Rovira i Virgili. Before starting my PhD studies, I began working at the laboratory with Carles Pairot and Ruben Mondéjar. Since then until nowadays I continued making good friends, like Heliodoro Tejedor, Marc Sànchez, Gerard París, Marcel Arrufat, Lluís Pàmies, Marc Espelt, and all the rest. I want to thank in particular to Marc Sànchez for our brainstormings, which have resulted very interesting and useful for each other thesis.

I thank to Prof. Alberto Montresor his friendship and the opportunity he provided me of making a research stay with him at Università degli Studi di Trento. Thanks also to Gianluca Ciccarelly and Alessandro Russo for their friendship during my stay at Trento. I want to thank to Juan Antonio Martínez his friendship and his hospitality all the times I have been at Murcia.

I want to thank to my family for their inconditional support all the time, from the very beginning of my university degree to this dissertation. I also thank all the support and love from my girlfriend Barbarakay Cisterna. Thank her for her patience along all these years. I would like to thank also the Cisterna's family, that even in the distance they have been thinking on me.

I would like to thank to all the people that have believed on me, that have shared their time and their friendship with me along all these years.

I finally thank to Universitat Rovira i Virgili and to the Spanish Ministry of Education and Science, under the FPU National Program, ref. AP-2006-04166, which supported this work.

Thanks to make it possible.

Jordi Pujol Ahulló

November 2009

Agradecimientos

Quiero agradecer sinceramente a mis directores de tesis, al Prof. Pedro García López y al Prof. Antonio F. Gómez Skarmeta, todo su tiempo y atención. Además, les quiero agradecer que me hayan dado total libertad para escoger el campo de investigación, después de lo cual me han ofrecido su total soporte y me han animado hasta la finalización de esta mi tesis. Ambos se caracterizan por ser extramadamente buenas personas y unos excelentes investigadores, cosa que ha sido un factor muy importante durante la evolución de mi investigación.

Quisiera también agradecer a todos mis amigos del Laboratorio de Arquitectura y Servicios Telemáticos (AST) de la Universitat Rovira i Virgili. Antes de empezar mis estudios de doctorado, empecé a trabajar en el mismo laboratorio con Carles Pairot y Rubén Mondéjar. Desde entonces he continuado teniendo muy buenos amigos, como Heliodoro Tejedor, Marc Sànchez, Gerard París, Marcel Arrufat, Lluís Pàmies, Marc Espelt y todos los demás, que por brevedad, no listo en estas líneas. En particular, quiero agradecer a Marc Sànchez por nuestras reuniones y brainstormings, que sin duda resultaron muy interesantes, a la vez que fructíferos, para nuestras correspondientes tesis.

También quiero hacer llegar mis agradecimientos al Prof. Alberto Montresor por su amistad y la oportunidad que me dió para realizar la estancia de investigación con él en la Università degli Studi di Trento, Italia. Gracias también a Gianluca Ciccarelly y a Alessandro Russo por su amistad durante mi estancia en Trento. Del mismo modo, quiero agradecer a Juan Antonio Martínez su amistad y su hospitalidad durante todas las veces que estuve en Murcia. Todo ello demuestra lo gran persona que son.

Deseo agradecer a toda mi familia su soporte incondicional durante todo este tiempo, desde el inicio de mis estudios en la universidad hasta la conclusión de mi doctorado. También quiero agradecer de todo corazón el soporte y el cariño que me ha dado mi novia Barbarakay Cisterna, así como también su paciencia durante todos estos años. Quiero agradecer también a la familia Cisterna que, aun en la distancia, hayan estado pensando en mi y hayan querido mi bien.

Agradezco a toda la gente que ha creído en mi, que ha compartido su tiempo conmigo, así como su amistad, durante todos estos años.

Finalmente, quiero dar gracias a la Universitat Rovira i Virgili y al Ministerio de Educación y Ciencia, bajo el programa nacional FPU, ref. AP-2006-04166, que han dado soporte a este trabajo.

Gracias a todos par hacer esto posible.

Jordi Pujol Ahulló

Noviembre 2009

Abstract

Peer-to-peer (P2P) networks are broadly classified into two main categories: unstructured and structured. Unstructured P2P networks (UPNs) were the first kind to appear and allow a great flexibility on user dynamicty, namely churn, whereas the data search is flooded. This search mechanism is inefficient and motivated the introduction of the structured P2P networks (SPNs). This new category organizes nodes in a proper way that guarantees a great lookup time efficiency and ensures that, if the piece of data exists, it is found.

The most relevant implementation of the structured peer-to-peer networks (SPNs) is constituted by distributed hash tables (DHTs). This implementation provides the same functionality than a traditional hash table, where buckets consist actually of the interconnected nodes. DHTs are mainly characterized by providing the pair of functions $put(key, value)$ and $value \leftarrow get(key)$. Above all, P2P networks are very attractive because they provide very interesting properties, like descentralization, by the use of computing resources at the edges of Internet; and system scalability by service distribution.

At a first glance this functionality could seem sufficient for a wide range of applications, but actually end-user applications require of enhanced services, such as high-level queries (e.g., range queries or top-K queries), as well as content distribution services (e.g., publish/subscribe or application level multicast (ALM)). Since the very beginning of SPNs, this kind of networks has been very permeable and has been receiving an enormous effort to improve SPNs by introducing all these services.

Nevertheless, a common factor among all those approaches is that they characterise a specific solution for a particular targeted system. The lack of genericity in these solutions make them probably efficient but non portable to other systems, and so the services constructed over them. We do believe, instead, that a SPN-generic solution is feasible. Another common factor is that SPNs use a number-based uni-dimensional keyspace to identify both nodes and data pieces, whilst the data domain of end-user applications can be of any kind, most of the times multi-dimensional (e.g., list of keywords for Information Retrieval or the pair latitude and longitude in location-based

services). Each approach adopts a particular solution for the targeted system, and not all the times a multi-dimensional scenario is reckoned with.

To this end, we propose a distributed framework of services for end-user applications. This framework is constructed by considering only the most common characteristics of existing SPNs, in order to guarantee at a high level its portability among SPNs. Having in mind this generic model of SPN, we define a set of distributed algorithms which will define the main rules on how high-level services should be resolved. This permits the flexibility of adding new services to the framework dynamically. Because data domains of end-user application will probably not coincide with the keyspace of the underlying SPN, we define an adaptation model for multi-dimensional data domains to the keyspace.

The benefits of our approach are clear: Our framework can be easily deployed over existing SPNs, guaranteeing the portability of a critical mass of services and end-user applications; Several services can be added to the framework, so new rich services are available for existing applications and will allow the construction of even more complex applications over our framework; Applications do not need to deal with data conversions when employing our services (the adaptation module deals with it transparently to services and applications).

We prove that our approach is feasible by defining three service modules. Firstly, a range query module defines the how to deal with range queries into our framework. Secondly, a geographical information service module allows to applications to find information by its location. Lastly, a publish/subscribe module provides to applications a way of distributing content through participant entities. A series of simulations prove their efficiency compared to the most related existing solutions.

Resumen

Las redes peer-to-peer (P2P) han sido clasificadas generalmente en dos tipos: desestructuradas y estructuradas. Las redes P2P desestructuradas fueron las primeras en aparecer y permiten una gran flexibilidad en cuanto a la dinamicidad de los usuarios se refiere. En estos sistemas las búsquedas de información se realizaban mediante la técnica de inundación. Este mecanismo de búsqueda es ineficiente y eso motivó la aparición de las redes P2P estructuradas. Esta nueva categoría de redes organiza los nodos en un modo adecuado para garantizar una alta eficiencia en el tiempo de búsqueda. Además, garantiza que si una información existe, el sistema la encuentra, propiedad que no se cumplía con la redes P2P desestructuradas.

La implementación más relevante dentro de las redes P2P estructuradas es la que constituyen las tablas de hash distribuidas (DHTs del término anglosajón "distributed hash tables"). Esta implementación provee la misma funcionalidad que una tabla de hash tradicional, en donde los buckets consisten en este caso en los nodos interconectados en la red P2P. Las DHTs, como la hacen las tablas de hash tradicionales, se caracterizan principalmente por proveer un par de funciones "$put(key, value)$" y "$valor \leftarrow get(key)$", y que se resuelven de forma eficiente. Todo esto permite demostrar que las redes P2P resultan muy atractivas. Por ejemplo, éstas garantizan la descentralización de los servicios, evitando así problemas congénitos del modelo cliente-servidor -como son el efecto cuello de botella, o el fallo del sistema en un sólo punto. Para ello se utilizan los recursos existentes en los nodos que participan en la red. Otra propiedad muy interesante es la consiguiente escalabilidad del sistema, debido precisamente a que la infraestructura del sistema ha sido distribuida.

Aunque a simple vista la funcionalidad de las DHTs pueda parecer suficiente para un amplio rango de aplicaciones, de hecho, las aplicaciones necesitan de otro tipo de servicios más avanzados, como son las búsquedas de alto nivel (por ejemplo, búsquedas de rango o búsquedas top-K), así como también técnicas de distribución de contenidos (por ejemplo, servicios de publicación/suscripción o multicast a nivel aplicación). Por ello, desde la aparición de las redes P2P estructuradas allá por el 2001, este tipo de redes han sido muy permeables y han recibido una gran atención para mejorarlas mediante la inclusión y desarrollo de este tipo de servicios.

No obstante, un factor común en todas las aproximaciones existentes es que sólo ofrecen una solución particular para un escenario específico. La falta de genericidad en estas soluciones las hacen probablemente eficientes para el caso para el que fueron diseñadas, pero no ofrecen portabilidad hacia otros sistemas y escenarios, y por lo tanto, lo mismo ocurre con los servicios que estos sistemas incorporan. En cambio, nosotros realmente creemos que una solución genérica con respecto a la red P2P estructurada es factible. No obstante, otro factor común en todas estas redes es que utilizan un espacio de claves uni-dimensional para identificar a los nodos y a los datos guardados en el sistema. Esto difiere del dominio de datos que utilizan las aplicaciones, puesto que la mayoría de las veces son complejos y multi-dimensionales (por ejemplo, una lista de palabras clave para un sistema de Recuperación de Información, o el par (latitud, longitud) en un servicio de localización geográfica). Es por ello que cada solución adopta una cierta aproximación para el escenario planteado, y, aun así, no siempre se da soporte para un dominio de datos de aplicación multi-dimensional.

Nosotros proponemos un marco de trabajo distribuido para el hospedaje de servicios, que a su vez ofrecerán sus funcionalidades a aplicaciones de usuario. Este marco de trabajo se construye considerando sólo las características más comunes en las redes P2P estructuradas. Con ello se quiere garantizar con alta probabilidad la portabilidad de nuestro marco de trabajo (y sus servicios) entre diferentes redes P2P estructuradas. El mecanismo que nos permitirá alcanzar tales objetivos tiene dos principales componentes. Primeramente, definimos unos algoritmos distribuidos sólo teniendo en cuenta las características comunes de las redes P2P estructuradas, por lo que resultan genéricos y aplicables en cualquiera de ellos. En segundo y último lugar, definimos un modelo de adaptación de los datos multi-dimensionales de aplicación al espacio de claves de las redes P2P estructuradas. Con todo ello, además, se abstrae y se facilita el desarrollo de nuevos servicios para nuestro marco de trabajo, así como también se permite que nuevos servicios se vayan añadiendo a él dinámicamente.

Para demostrar que nuestro marco de trabajo es factible, definimos tres módulos de servicios. El primero define cómo solucionar búsquedas de rango utilizando nuestro marco de trabajo. En segundo lugar, desarrollamos un servicio de información geográfica que permite, a las aplicaciones que lo utilicen, encontrar objetos relevantes por medio de su localización. Por último, proveemos un módulo de servicios de publicación/suscripción, que permite distribuir contenidos en un sistema distribuido a todos aquellos participantes que estén interesados. Validamos todos estos servicios, así como nuestro mecanismo de adaptación de datos por medio de simulaciones en

cada uno de los escenarios. Los resultados demuestran que son eficientes, en comparación con otros sistemas estrechamente ligados a cada escenario, así como que nuestro marco de trabajo funciona con un buen rendimiento sobre diferentes redes P2P estructuradas.

En particular, para facilitar que nuestro marco de trabajo pueda utilizar el sustrato P2P como medio de comunicación, sólo utilizamos un conjunto mínimo de funciones ya provistas o fácilmente suministrables por las redes P2P estructuradas. Estas funciones son *sendMessage*, *getLinks* and *deliverMessage*. Desde un punto de vista arquitectónico, el marco de trabajo está formado por diferentes módulos, donde cada uno es responsable de dar solución a diferentes problemas. A continuación los detallamos para mayor detalle.

Módulo de adaptación de datos. Éste es el componente clave de nuestro marco de trabajo. Cabe destacar que las redes P2P estructuradas que se consideran en esta tesis, tienen un espacio de claves unidimensional. Como consecuencia, *los dominios de datos multi-dimensionales de las aplicaciones no son directamente soportados* por las redes P2P estructuradas. Este módulo se erige para realizar la transformación de los dominios de datos multi-dimensionales a una única representación. Tal dominio de representación coincide con el espacio de claves de la redes P2P estructuradas, por lo que de este modo, los dominios de datos de aplicación son transparente y elegantemente soportados por la red P2P subyacente.

Para realizar tal transformación, hemos diseñado una técnica de reducción dimensional, llamada Bit Mapping (BM). Además, debido a que nuestro marco de trabajo tiene que soportar diferentes servicios y aplicaciones al mismo tiempo, BM es genérica a cualquier dominio de datos de aplicación. El diseño y posterior desarrollo de la función BM fueron motivados porque las funciones existentes no se adecuaban a los requerimientos de nuestra infraestructura, tales como *balanceo de carga de datos y encaminamiento*, mientras que las operaciones garantiza que se resuelvan *eficientemente*.

Además, las operaciones distribuidas (como por ejemplo, inserción de datos o operaciones basadas en rango) son resueltas por los servicios instalados en nuestro marco de trabajo. En otras palabras, diseñamos los algoritmos necesarios para dar soporte a operaciones complejas dentro de nuestra infraestructura. Para ello, no construimos nuevas redes P2P, sino que nos basamos en la red P2P estructurada subyacente.

Estos algoritmos son formalmente evaluados por medio de un análisis teórico, así como también por medio de una evaluación por simulación (en cada uno de los casos presentados en los demás módulos). Nuestros algoritmos de encaminamiento

distribuidos demuestran que son eficientes en una gran variedad de escenarios, y escalan en el número de nodos y en el número de dimensiones del dominio de datos de las aplicaciones de usuario.

Módulo de gestión de datos. Este módulo presenta dos casos de uso de despliegue de servicios en nuestro marco de trabajo. El primero es el diseño, desarrollo y evaluación de un servicio que provee búsquedas de rango de datos multi-dimensionales. Los datos multi-dimensionales se almacenan en la red P2P. Para ello, se usa la función BM para producir las correspondientes claves. Los nodos responsables de tales claves almacenan los datos.

Lo que ofrece este módulo entonces es en primer lugar un servicio de búsquedas de rango de datos multi-dimensionales. Su evaluación de rendimiento demuestra la eficiencia no sólo de la función de adaptación BM, sino también de los algoritmos descritos en el anterior módulo de adaptación de datos. Ambos conjuntamente muestran un excelente balanceo de carga de datos y de encaminamiento en todo el sistema, comparado con otras soluciones existentes.

Además del servicio anterior, este módulo también provee un servicio de información geográfica. Caracterizamos el problema en cuestión e identificamos todos los problemas que hay que salvar, tales como *datos multi-dimensionales* o cómo *localizar los datos por su zona geográfica*. Para superar los retos identificados, proponemos que tanto los datos como los nodos se organicen en grupos según su proximidad geográfica. Para ello, adaptamos ligeramente la función BM para poder utilizar la información geográfica como parte en la fase de adaptación de los objetos a almacenar y buscar.

A parte de todo esto, también diseñamos una red P2P estructurada jerárquica particular, que se beneficia de la propia agrupación geográficamente de nodos y datos. Los datos, como en el caso anterior, se almacenan en los nodos responsables de las claves producidas por la adaptación de los datos. A partir de ahí, presentamos cómo se pueden desarrollar búsquedas exactas, búsquedas geográficas y una novedosa búsqueda llamada geocast. Aun cuando el dominio de datos de aplicación es multi-dimensional (localización y descripción semántica), nuestra propuesta es capaz de resolver todos estos tipos de búsquedas muy eficientemente. Cabe destacar que las búsquedas geocast recuperan la información del mismo tipo (por descripción semántica) pero de diferentes lugares geográficos a la vez a un muy bajo coste, comparado con otros sistemas existentes. Para concluir, nuestra agrupación de datos y nodos también permiten aventajarnos frente a otras propuestas, puesto que garantizan localidad en

el camino, además de localidad de datos, lo que lleva a que la solución sea eficiente y escalable.

Módulo de distribución de contenidos. No sólo nuestro marco de trabajo permite gestionar información en un entorno distribuido, sino que también distribuir contenidos. En nuestro caso, este módulo provee servicios de publicación/suscripción basados en el contenido. Utilizando la misma idea que antes, adaptamos eventos y suscripciones al espacio de claves de la red P2P estructurada por medio de la función BM. A partir de ahí, demostramos que si un evento E es capturado por una suscripción S, la correspondiente clave de la transformación de E aparecerá en el conjunto de claves de la adaptación de la suscripción S.

Teniendo esta propiedad en mente y que nos basamos en las capacidades de encaminamiento de la red P2P subyacente, la suscripciones se gestionan del siguiente modo. Empleamos el modelo "de punto de encuentro.º rendezvous para hacer coincidir los eventos con las suscripciones correspondientes en un escenario distribuido. Para ello, las suscripciones se almacenan en todos aquellos nodos responsables de sus claves de transformación. Entonces, los eventos (que se transforman a una sola clave) se envían a sus nodos responsables. En cualquier caso, utilizamos la función de transformación BM y los algoritmos de encaminamiento del módulo de adaptación de datos. Una vez los nodos responsables reciben un nuevo evento, los nodos escogen aquellas suscripciones almacenadas localmente que seleccionan al evento recibido. A partir de este momento, el nodo en cuestión inicia el proceso de notificación del evento a todos los nodos suscriptores seleccionados en el paso anterior.

El conjunto de operaciones siempre se realizan utilizando solamente información de estado local a los nodos. Mediante simulaciones demostramos que nuestra propuesta provee satisfactoriamente los servicios de publicación/suscripción. Además, el servicio demuestra sus grandes cualidades, tales como balanceo de carga de datos y de encaminamiento o escalabilidad, en términos de número de nodos y de número de dimensiones del dominio de datos de aplicación.

Por lo tanto, los beneficios de nuestra aproximación son claros: (i) Nuestro marco de trabajo puede ser fácilmente desplegando sobre las redes P2P estructuradas existentes, garantizando de este modo la portabilidad de una masa crítica de servicios y aplicaciones de usuario; (ii) Nuestro marco de trabajo puede hospedar múltiples servicios, por lo que las aplicaciones que usen nuestro marco de trabajo se pueden beneficiar de todos ellos y, de esto modo, permitimos que se puedan construir aplicaciones más complejas sobre nuestra infraestructura; (iii) Las aplicaciones no necesitan

realizar ninguna conversión de datos (nuestro módulo de adaptación lo gestiona de forma transparente tanto a servicios como a aplicaciones), así como tampoco necesita conocer las propiedades específicas del sustrato P2P que se esté utilizando (nuestros algoritmos distribuidos ya se encargan de completar las operaciones de forma eficiente).

Contents

List of Figures

List of Tables

1

Introduction

Advanced services like complex queries (e.g., range queries or spatial searches) and content distribution services (e.g., publish/subscribe systems) are of great interest and necessary for lots of modern end-user applications. For instance, Flikr [1] allow users to upload and share images. Further, it allows users to *geotag* images, by which images become labelled with their geographical location and can be seen in Flikr Maps [2]. This is an example of spatial search, where the images from a certain location are selected from. In addition, users are kept up to date by the Flikr Keep in Touch service [3]. This enables users to know what are doing the contacts that they have. This is clearly a content distribution service, where after recording users their interests upon other users, they are notified with the last changes and news from their contacts.

Flikr is a centralized service operating in a client/server mode though. It requires an enormous investment (in equipment) to support the huge amount of data and traffic that the public service causes. An alternate solution could turn Flikr into a distributed service. To this end, let us suppose that Flikr interconnects users by sharing their images and related information from their Flikr's desktop application, users' weblog or image galleries from their personal web sites [4]. This distributed solution would strive after cooperating and collaborating users' applications and web services to provide the same service as in the centralized approach. Indeed, the aforementioned advanced services should be also supported by the distributed version of Flikr. These sophisticated services are non-trivially resolvable in a distributed, large scale fashion. A very interesting and exciting approach comprises the use of the peer-to-peer computing techniques to address these complex data management and content distribution services.

Peer-to-peer (P2P) networks gained their popularity with the introduction of distributed file-sharing applications like the first Napster [5] and later Gnutella [6]. This kind of networks provided an alternative to the traditional client/server communication model. The P2P computing provided rattling good properties, like no central point of failure, no service bottleneks, all this by decentralizing the service among participating nodes. That new technology appeared as the first of taking advantage of

available resources at the edges of Internet (like free hard disk space, available computation cycles or unused main memory) for a system's common goal. But this brought up the necessity of a coordination mechanism between nodes.

Lots of P2P protocols provided several ways of node inter-communication and coordination. According to the way nodes are interconnected, they are roughly classified into two categories: unstructured and structured. Unstructured peer-to-peer networks (UPNs) (e.g., Gnutella [6]) build a networked system whose nodes are arbitrarily connected between each other. This model of network allows users to store an arbitrary set of objects (i.e., files and information of any kind) into their nodes, being shared among all the rest of users. Given this network structure, searches are flooded with a time-to-live (TTL) counter between participants. The TTL's goal is to prevent the search from being flooded indefinitely. This approach does not provide a unique view of the system though. Depending on the user's node location into the networked system, a user can encounter a set of objects highly probable different from a user in another system location. Another drawback is the high cost of a search resolution, since the query is broadcasted to all nodes.

It is clear, therefore, that this network model does not guarantee the search correctness. That is, if an object exists, it is found. Structured peer-to-peer networks (SPNs) appeared to dodge this obstacle (e.g., Chord [7], Pastry [8], Tapestry [9], Bamboo [10], CAN [11], Koorde [12], Symphony [13], P-Grid [14], Kademlia [15]). SPN nodes are connected in a particular way, so that every node can find a path to any other node at a relatively low cost. To do so, every node is labelled with a value, namely *nodeID*, which identifies a node uniquely within the system. In most of the SPNs, nodeIDs are drawn from a keyspace of a single numerical value of the form $[0..2^m)$, where m is the bit precision of the numeric value [7, 8, 9, 12, 14, 15] (except those derivative from CAN [11], that use a numerical D-dimensional keyspace, defining a D-dimensional torus).

Distributed hash tables (DHTs) embodies the set of SPNs. This is why most of the times SPNs are generically called DHTs. In particular, a DHT is a distributed data structure that provides the same functionality than a traditional hash table (i.e., $put(key, value)$, $value \leftarrow get(key)$). The main characteristic of a DHT is that nodes play the role of buckets of the hash tables. This way, all participating nodes share the responsibility of storing and retrieving users' information. To do so, a *key* is produced from the user's information. The keyspace for these keys is the same than for the nodeIDs, so that a key uniquely identifies the node that will be responsible of it. In most of the systems, a message can be transmitted between two nodes solely in a

logarithmic number of hops, with respect to the number of nodes into the system. Therefore, operations in SPNs are cheaper than in UPNs. All these properties motivate to take only SPNs in consideration in this thesis.

A key property of a P2P setting is the dynamicity of users, namely churn, where participants can enter and leave the network at any time. This contextual characteristic of the P2P systems leans over the protocol design the responsibility of guaranteeing important system's attributes, like service reliability or data availability, even in the presence of churn. To do so, systems improve their protocols with caching and replication techniques [16, 17, 18]. For instance, replicating objects into several specific nodes will likely ensure a high degree of data availability, even if the node owning those objects fails. In such a case, new operations would be directed to nodes with object replicas. Since these techniques are intrinsically specific of the targeted P2P system, we do not consider caching or replication mechanisms in our thesis, but as an indivisible part of the P2P protocol.

1.1 Topic and Motivation of this Thesis

Basic SPN services (i.e., *put/get*) are not enough for up to date complex applications, like a distributed Flikr, document indexing [19], geographical information services [20], or application-level multicast [21] and publish/subscribe services [22]. These modern applications necessitate of high-level functions that make easier to develop such systems. To do so, a candidate P2P substrate must deal with the following concerns: *service correctness* and *efficiency*, *high-level service provisioning* and *complex data domain support*.

- **Service correctness.** Correctness in a service is fundamental. Since SPNs construct a global directory service (what means that for a given *key*, there exists a unique node responsible for it[1]), this guarantees correctness to any service built upon SPNs.

- **Service efficiency.** This property tells whether communications within P2P network are efficient (for instance, with a low operation latency). Most of the SPNs ensures a logarithmic cost in the worst case ([7, 8, 9, 13, 23]) in number of visited nodes, with also a logarithmic number of entries in the routing table. This makes the solution efficient given their distributed nature.

[1]Conversely, if more than one node is responsible for *key*, such system was designed structurally to support replication, or such system incorporated replication techniques. Since we do not consider replication techniques in this thesis, we reckon all nodes in the replica set as a single one.

- **High-level service provisioning.** On the surface, SPNs seem to provide the necessary functionality for modern applications. But there are lots of applications that pose more complex challenges than a simple *put/get* to SPNs in order to be supported. Services focused on either distributed data management or content distribution are roughly the two main families. For instance, *k*-nearest neighbor (kNN) queries are an elementary part of document indexing applications [19]; range or window queries are the basis for geographical information services [20]; publish/subscribe services for many-to-many communication, either in the topic-based [21] or content-based [22] model.

- **Complex data domain support.** Application data domains are uneven from each other. For instance, the data domain of a document indexing application is the set of keywords with which documents are described [24]; image databases use real-valued feature vectors of lenght F to identify images, where each feature characterise a property from the image [25]. Since most of the SPNs, though, utilize a unidimensional numerical keyspace, this poses a structural challenge and must be tackled within the proposed solution.

Existing solutions have adopted one out of these two most common approaches: (i) **constructing an ad-hoc P2P infrastructure** supporting explicitly the application data domain [20, 26, 27, 28, 29], or (ii) **adapting the data domain** in order to map a multi-dimensional data object to the unidimensional key of the SPN [30, 31, 32]. This operation is performed by *linearization functions*, like Space-Filling Curves (SFC) [33] (e.g., Z-Curve [34], Hilbert curve [35], XZ-Ordering [36]), order-preserving hash functions (OPHF) or locality-preserving hash functions (LPHF). According to the specific context, each solution selected one kind of function to fulfill the application requisites (e.g., systems using SFCs [32], OPHF [31] or LPHF [30]). However, all these solutions are designed to be efficient in the specified context, what brings a set of shortages to light:

- **Maintenance.** A common mechanism to design new services is to build a particular P2P network (namely overlay), which facilitates the deployment of such services. These new overlays are very often constructed over already existing overlays (e.g., Scribe is deployed onto Pastry, or Bayeux is settled up on Tapestry). Since the P2P networks are dynamic, where nodes can leave and enter the network at any time, P2P protocols must consider this issue. This produces signaling traffic for any (overlay of) overlay, since each overlay has to maintain

the overlay's properties, such as network connectivity or fan-out. This effort is invested at each (overlay of) overlay and, hence, this results in a duplicated, non-trivial signaling traffic. Moreover, it is not fiddling how an overlay $\mathcal{O}$ can deal with churn if the underlying overlay $\mathcal{Q}$ (which supports $\mathcal{O}$) is also suffering from churn.

- **Genericity.** Most of the existing P2P solutions provide interesting services and build end-user applications employing specific P2P infrastructures. For instance, a new service can be provided by means of an overlay construction [21], or it can be designed to operate on a specific P2P infrastructure [30]. Therefore, this lack of genericity on the P2P infrastructure prevents the service from being deployed onto other P2P systems. To this end, a Common API appeared in order to define the common set of functions that high-level services and applications would require for its construction [37]. Even though several systems implemented the Common API (e.g., FreePastry[38], a free implementation of Pastry [8]; Bamboo [10], a free implementation of a DHT; PlanetSim [39], a P2P network simulator which includes Chord and Symphony), their implementation consist of particular variations or interpretations of the Common API, so that services and applications built onto those systems are actually not fully generic.

- **Portability.** As a result of the above issue, the enormous budget of interesting and available services and applications becomes unusable for other P2P systems. This abstraction between the service/application logic and the P2P infrastructure would reward on cost reduction of its development process, given that the service would successfully work on a wide range of P2P systems.

- **Multiplicity of services.** Another common characteristic of most of the proposed solutions is that the service and the P2P infrastructure is delivered into a single building block. This way, this monolithic design precludes several services from being deployed in a single P2P infrastructure. Very few proposals (such as SkipIndex [28]) were structurally designed to support multiple services. However, they fail in providing genericity, for instance.

Instead, the conducting motivation of this thesis is the construction of a generic framework. We consider that the facets of the term *genericity* should be twofold:

- **Extensible framework.** Any kind of service developed into this framework could be added at any time, so that new services would be available for end-

user applications. This way, a service designed to work onto this framework could be portable between any P2P systems that use this framework.

- **SPN-generic framework.** Not only should services be portable, but also the framework should. That is, the framework should be capable of being deployed in most of the SPNs. This is a necessary property to ensure service portability and reuse, since they would be designed for this framework.

Distributed end-user applications demand, broadly speaking, two main types of services: data management and content distribution services. The former can be seen as a distributed service of data storage and/or indexing, where participants can store information of any kind and perform complex data searches. Image database or geographical information systems are some examples. The latter provide a many-to-many communication model, by which it makes aware of new pieces of information to an arbitrary number of nodes. Application-level multicast and publish/subscribe systems embody this kind of services.

In addition, both kind of services must tackle the definition of a distributed data structure and the related algorithms to perform an expected set of operations with it. This duality presents a trade-off that is not new, but it raises in the field of software engineering: how complex the data structure is *vs.* how complicated the algorithms become for accessing the given data structure.

This thesis considers this common scenario, which should serve as a basis for the definition of the core infrastructure of the framework. If we provide a common data structure with the framework to both data management and content distribution services, this would facilitate the framework construction as well as it would unify the accessing model to nodes in the SPN. In consequence, service implementation and deployment would be easier.

The last, but not the least, issue that this thesis addresses is the data characterization. Each end-user application works with a particular data domain. For instance, the pair $\{longitude, latitude\}$ is the main data entity for a geographical information system; or a vector of keywords identifies a document into an Information Retrieval system. This data domain is most of the times different from the keyspace of the SPN. The proposed framework should tackle the adaptation of the data domain to the keyspace too, as existing solutions do. Space-Filling Curves or order-preserving hash functions are examples of such transformation techniques ([33, 35, 36, 40, 41]). As to guarantee the genericity of the framework, the adaptation mechanism should face not only the data type (e.g., strings or numbers), but also the data multi-dimensionality (e.g., the

pair $\{longitude, latitude\}$ is a data object, which is a bi-dimensional object, formed by two single elements).

1.2 Contributions of this Thesis

We do believe that a generic framework is possible, a framework that could be deployed over most of SPNs, as well as support several services necessary for end-user applications. We begin with a study on the state of the art on the services our framework will support. The idea behind that is to discover the infrastructure requirements of such services, which are considered in the design of our framework. Afterwards, we describe the work of this thesis as modules, each of which deals with a significant part of the proposed framework. The first module addresses core functionalities of the framework, and the rest of the modules depict use cases where some service is added to the framework.

Analysis of the state of the art. Before starting on the design of our framework, we make a concise study on the state of the art, in particular on the fields of the services we aim at providing with our framework.

- **First contribution.** We realize the analysis on the state-of-the-art on the fields of similarity queries (range queries, k-nearest neighbors queries, spatial queries) and publish/subscribe services (topic- and content-based paradigms). To do so, we state a common evaluation framework on each field with up to 10 quantitative and qualitative properties. We conclude this analysis with some open issues that we address in this work.

Data adaptation module. This module is the core component of the proposed framework, since it deals with the data domain adaptation to the keyspace, as well as proposes generic algorithms that can be used for high-level services.

- **Second contribution.** We provide a mapping function, namely BM, which adapts any data domain to the keyspace. The adaptation mechanism is deterministic, so that, given the same inputs, any node would produce the same key or keys within the keyspace. This is a necessary property, since this determinism meets the rendezvous model that SPNs obey.

- **Third contribution.** In order to allow specific services to run with this framework, we provide a set of algorithms that work over most of SPNs. To do so, we consider a minimum subset of their common characteristics, with which we implement generic algorithms for range-based operations.

Data management module. This module exemplifies the use of the adaptation module to provide data management services. We have focused our research effort on two out of the most demanded services: range query and geographical location services.

- **Fourth contribution.** We introduce SQS, a module service capable of performing multi-dimensional range queries. To do so, this module indexes the information into the SPN in a proper way using the adaptation module, so that the proposed search algorithms can easily deal with range queries.

- **Fifth contribution.** Geophony is our proposal to tackle the necessities of geographical information systems: geo-localization. We suggest a particular node organization within a SPN, over which we realize the indexation of geographical data by applying a smooth variation to the mapping function. The same proposed search algorithms support geographical lookups though. In addition, we provide a new kind of lookup operation, called geocast, which is able to discover interesting information at different geographical granularities (for instance, regional, national or continental granularities).

Content distribution module. One of the most common exigencies of end-user applications is the possibility to announce new pieces of information to several or even thousands of users. Application-level multicast and publish/subscribe services are clear examples in this context.

- **Sixth contribution.** We propose CAPS, a multi-dimensional content-based publish/subscribe service. We construct the service by employing the proposed data adaptation module as is, and including extra algorithms to distribute the notifications to all interested participants.

1.3 Outline of this Dissertation

We detail in the following lines the structure of this dissertation. It is easy to note that its organization mainly follows the set of modules presented in the section before.

Chapter 2: Background and State of the Art. We firstly outline some nomenclature and the common properties related to SPNs. They are used along the development of this thesis. In addition, this chapter includes an overview of a hierarchical

SPN, namely Cyclone [42], which is used as a substrate in several evaluation scenarios of our framework. In the last part of this chapter, we perform an analysis of works in the state-of-the-art, in all the fields this thesis is involved into. This includes adaptation techniques (mainly linearization functions), high-level search services (range queries or k-nearest neighbor queries), geographical location services, as well as publish/subscribe systems.

Chapter 3: A Framework for Developing Application-level Services. In this chapter we introduce our adaptation module, including our BM function, which is able to adapt multi-dimensional data domains to specific keyspaces, as well as suitable algorithms based on the properties of our adaptation technique to perform complex range-based operations.

Chapter 4: Multi-dimensional data management. We introduce in this chapter SQS and Geophony. SQS is our service module that implements range queries into our proposed framework, using the tools provided by the adaptation module. We also propose an underlying SPN where to settle our framework. Conversely, Geophony consists of not only the service module for geographical location, but also a proposal of a suitable P2P infrastructure where the framework should be set up to deal more efficientlty with this kind of services. As a consequence of our approach, Geophony also contributes with a new kind of queries called geocast. Both service modules are then validated through significative simulation settings, and their results establish the soundness of our approach.

Chapter 5: Content distribution capabilities. This chapter introduces the service module for publish/subscribe services, namely CAPS. CAPS is a multi-dimensional content-based publish/subscribe service. To develop it, we base the subscription and notification functions in the adaptation module, incorporating a new algorithm for the distribution of new notifications. Through simulation results, we corroborate the effectiveness of CAPS and its efficiency.

Chapter 6: Conclusions and future work. This last chapter serves to expound the main conclusions on the work of this dissertation, as well as to expose some future research lines.

1.4 Selected Publications

This thesis is based on the following publications:

- **Jordi Pujol Ahulló**, Pedro García López, Marc Sànchez Artigas and Antonio F. Gómez Skarmeta. *SQS: Similarity Query Scheme for Peer-to-Peer Databases*. In Proceedings of 12th IEEE Symposium on Computers and Communications (ISCC'07). Aveiro, Portugal, July 1-4 2007, Pages 1107-1112. ISBN: 978-1-4244-1521-2. ISSN: 1530-1346. IEEE Computer Society. Los Alamitos, CA, USA

- **Jordi Pujol Ahulló**, Pedro García López and Antonio F. Gómez Skarmeta. *LightPS: Lightweight Content-based Publish/Subscribe for Peer-to-Peer Systems*. In Proceedings of 2nd International Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC-2008), held in conjunction with International Conference on Complex, Intelligent and Software Intensive Systems (CISIS-2008). Barcelona, Spain, March 4-7 2008, Pages 342-347. ISBN: 978-0-7695-3109-0. IEEE Computer Society. Los Alamitos, CA, USA.

- **Jordi Pujol Ahulló**, Pedro García López and Antonio F. Gómez Skarmeta. *CAPS: Content-bAsed Publish/Subscribe services for peer-to-peer systems*. In Proceedings of 2nd International Conference on Distributed Event-Based Systems (DEBS '08). Short Paper. Rome, Italy, 1-4 July 2008.

- **Jordi Pujol Ahulló**, Pedro García López, Marc Sànchez Artigas and Antonio F. Gómez Skarmeta. *Supporting Geographical Queries onto DHTs*. In Proceedings of 33rd IEEE Conference on Local Computer Networks (LCN '08). Montreal, Canada, 14-17 October 2008, Pages 435-442. ISBN: 978-1-4244-2413-9. ISSN: 0742-1303

- **Jordi Pujol Ahulló**, Pedro García López and Antonio F. Gómez Skarmeta. *Towards a Lightweight Content-based Publish/Subscribe Services for Peer-to-peer Systems*. Special Issue on Efficient Resource, Service and Data Models for Grid and P2P-Enabled Applications. International Journal of Grid and Utility Computing (IJGUC). To be published, January 2009. ISSN: 1741-8488 (Online), 1741-847X (Print).

- **Jordi Pujol-Ahulló** and Pedro García-López. Chapter: *Enhanced Lookup Queries for Large-Scale Distributed Data Structures*. Parallel Programming and Applications in Grid, P2P and Network-based Systems Series Advances In Parallel Computing. May 2009 , Pages 83-117. IOS Press. Amsterdam, The Netherlands.

2

Background and State of the Art

Structured peer-to-peer networks are a class of distributed data structures that are designed to provide *scalability by service distribution*. However, SPNs provided initially only exact-match queries in a scalable way. Exact-match queries consist of retrieving single values from the network (i.e., *put(key, value)*/*value←get(key)*) and providing operation correctness (i.e., in such a way that if the data exists, it is found).

Unfortunately, exact-match queries are not enough for modern high-level services, such as range queries. Broadly speaking, a range query is designed to retrieve all objects from the specified segment or range. A first, primitive approach to provide range queries would consists on flooding the whole network with the query, but it is not suitable in the large scale. On the contrary, one could think that high-level services (such as range queries) could be constructed through the use of exact-match queries. To explain how, let q be a query that needs to retrieve all objects in the range $[A..B] \subseteq \mathcal{I}$, where $\mathcal{I}$ is the SPN keyspace. The result set for query q would be $\mathcal{O}' = \{o \in \mathcal{O} | A \leq \mathcal{F}_{\mathcal{O}}(o) \leq B\}$, where $\mathcal{F}_{\mathcal{O}}$ is the mapping function that for a given object o provides a key $k \in \mathcal{I}$. Let us suppose that $||[A..B]|| = d$ consists of thousands of keys. A naive solution would be to ask for all individual keys $i \in [A..B]$. Nevertheless, this approach would turn very inefficient for different concerns. Firstly, the whole operation would involve thousands of individual *get(i)* queries. Such an effort is unfeasible, because of the waste of resources not only in the node that performs the query, but also in the cooperative nodes that would contribute their time and resources in the query resolution. Secondly, there is no precondition that ensures that for a given key i would exist (at least) an object o, which ensues in squandered system resources. Indeed, a node p could potentially be the responsible node for lots of individual keys i, so that lots of individual queries would target an overloaded p. It is clear, then, that these naive approaches do not suit for high-level services, but other kind of more efficient and (probably) complex solutions are demanded. Therefore, scalable high-level service provisioning on SPNs is challenging.

The services that modern distributed applications ask for, comprise two main families: *data management* and *content distribution* services. Actually, modern applications

manage user information, storing and indexing it throughout the system for global efficiency. But users are not only interested in using the application as a database system, but also for receiving notifications from other data sources (e.g, other users in applicatations like Facebook or Twitter). Therefore, users need to *be aware* whether some set of interesting data has recently appeared (e.g., news) or has been changed (e.g., added new photos or changes in users' profile). Clearly, the first kind of service allows users to *manage* their *information*, whilst the second one permits the system to *distribute* pieces of information to users *according to their interests*.

In particular, we focus on the following services, since they are tightly related to different parts on the work of this thesis. Range queries, k-nearest neighbor queries and spatial queries belong to the family of *similarity queries* and permit an enhanced data management. As the counterpart, application-level multicast and publish/subscribe services are the most common techniques to disseminate content among participants. This chapter builds a portrait of the state-of-the-art, by performing a comparison of the last and most remarkable works in the field of similarity queries and publish/subscribe services. To do so, we state an evaluation framework with which all systems are compared to. Examples of the analysed parameters within the evaluation framework are: overlay network topology, application domain, query correctness, completeness, storage and time efficiency and load balancing, to say the least. The goal, thus, is to provide an insight on peer-to-peer-enabled distributed algorithms that support similarity queries, as well as publish/subscribe solutions, in the large scale.

The rest of the chapter is structured as follows. We firstly state the necessary peer-to-peer background and notation for the development of this thesis. Several concerns are involved. Firstly, we detail the common properties from the majority of the SPNs (Section 2.1). Secondly, we also present a hierarchical SPN, Cyclone [42] at Section 2.1.2, which is employed as a substrate in several services of our framework. In addition, this chapter provides the notation set which will be used along this thesis. Afterwards, we deal separately firstly with similarity queries and secondly with publish/subscribe techniques. This is motivated for the specific characteristics of each scenario, imposing different requirements to the supporting system.

In Section 2.2 we introduce the data management services we will study. Afterwards, we describe in Section 2.3 how nodes cooperate by parallelizing operations, a common feature on distributed services. In Section 2.4 we analyse the parameters we include in the evaluation framework for the similarity queries. We then perform the comparison study of systems providing range queries (Section 2.5), k-nearest neighbors (Section 2.6) and spatial queries (Section 2.7).

Likewise, in Section 2.8 we present the content distribution services included in our analysis. In Section 2.9 we introduce the parallel computing abilities of the publish/subscribe algorithms. We then elaborate on the parameters that we include in the evaluation framework for the publish/subscribe techniques in Section 2.10. We continue with the comparison study of systems (in Section 2.11) following either the topic-based model (Section 2.11.1), or the content-based model (Section 2.11.2). This chapter closes with the concluding remarks in Section 2.13.

2.1 Peer-to-peer Networks

Unstructured Peer-to-Peer Networks (UPNs) appeared earlier to bring together edge resources. File sharing applications are predominant in this context, where Gnutella [6] embodies UPNs. They are called unstructured because links between nodes are established arbitrarily. Nevertheless, UPNs suffers of two main drawbacks: lack of query *correctness* and overhead on communication. The former can be explained as follows: suppose there is a file in the UPN and a user wants to retrieve it; if such a file is too far from the user, it might not to be found by the user query. The latter occurs because communication in UPN is performed mainly by flooding and causes a high amount of signaling traffic in the network. Hence, such networks typically have very poor search efficiency.

Structured Peer-to-Peer Networks (SPNs) appeared to overcome the problems appeared in UPNs. SPNs are to provide *correctness*, a key property for modern applications, as well as routing and time efficiency. SPNs are also broadly called distributed hash tables (DHTs) because most of them associate the data owner node by means of a consistent hashing, in an analogous way to that of traditional hash table's assignment of keys to a bucket. SPNs provide exact-match queries with correctness and applications can use the API *put(key, value)*/*value*←*get(key)* to access to the content.

Remark 2.1 (SPNs' correctness) *SPNs guarantee operation correctness. That is, given a key k, SPNs can always determine the responsible node p of k.*

2.1.1 Common Properties of Structured Peer-to-Peer Networks

Systems like Chord [7], Pastry [8], Tapestry [9], Kademlia [15], CAN [11] or Symphony [13] are clear examples of SPNs, and share a common set of properties. We elaborate on them in the following lines, based on the description appeared at [43]. Structured peer-to-peer networks can be defined by a a set of nodes $\mathcal{P}$ that cooperate together to provide access to a set of objects $\mathcal{O}$ (i.e., any kind of information being

stored by users). The access is provided by means of a keyspace $\mathcal{I}$ and the function $\mathcal{F}_{\mathcal{P}} : \mathcal{P} \longrightarrow \mathcal{I}$, which constructs the node identifier, and an adaptation function $\mathcal{F}_{\mathcal{O}} : \mathcal{O} \longrightarrow \mathcal{I}$, which builds a representative key for the given data object. This way, these functions establish the assignment of objects to peers. Even though systems like CAN and derivative ones employ a multidimensional keyspace $\mathcal{I}$, the majority of SPNs employ a single numerical dimension for its keyspace $\mathcal{I}$. This motivates to establish the following assumption without loss of generality:

Assumption 2.1 (Keyspace and ID) *Nodes in SPNs are uniquely identified within the system by a unidimensional node identifier, namely ID, drawn from the keyspace $\mathcal{I} = [0..2^m)$, where m specifies the keyspace precision.*

The function $\mathcal{F}_{\mathcal{P}} : \mathcal{P} \longrightarrow \mathcal{I}$ splits $\mathcal{I}$ into disjoint partitions. At any instant, the current set of identifiers defines the current set of partitions in which the identifier space has been divided into. Each node is the responsible for a distinct partition. By responsible node we mean that each node is the manager for all objects whose identifier is mapped into that partition according to the adaptation function $\mathcal{F}_{\mathcal{O}} : \mathcal{O} \longrightarrow \mathcal{I}$. In particular, the distributional properties of $\mathcal{F}_{\mathcal{O}}$ have a critical impact on load balancing. And this is a factor that we will consider and analyse within our generic framework.

Assumption 2.2 (Shared responsibility of the keyspace) *The keyspace responsibility is shared among all participating nodes. That is, for any $\mathcal{I}$'s disjoint partition I, there exists a node p responsible for I, according to the SPN's responsibility rule.*

Traditionally, SPNs maximized load balancing using a hash function to map both peers and objects into the keyspace. Cryptographic hash-functions like SHA1 are usually used to map arbitrary strings to 160-bit keys, which provide a consistent hashing of node identifiers and keys within the keyspace. Associated to the keyspace, SPNs define a distance metric, which is used to route greedily towards the node responsible for a key. Examples of distance metrics are the Euclidean distance adopted in CAN [11], the length of the common prefix adopted in Pastry [8], Tapestry [9], and Kademlia [15] (also known as XOR metric), or the clockwise distance between two ids on the circle $[0, 2^m)$ modulo 2^m, specified in Chord [7].

Assumption 2.3 (Information state) *Nodes in SPNs construct a routing table of a logarithmic degree, namely with $O(\log N)$ entries, where N is the number of nodes in the network.*

The distance metric together with SPN's greedy routing outperform to UPNs in the communication cost. To do so, SPNs usually construct a routing table of a logarithmic

length (i.e., small degree) according to N, the number of nodes in the system, with references to other nodes from within the network. With all these ingredients, SPNs reduce the communication cost down to a worst case $O(\log N)$ in number of visited nodes (i.e., small diameter). As a counterexample, we illustrate CAN. CAN place all nodes in a multi-dimensional torus, but with a constant small degree. For a d-dimensional torus, the routing table is $2d$ and the communication cost grows up to $O(N^{1/d})$. Nevertheless, a later eCAN [23] improved the routing table structure by adding short-cuts to far away nodes and improved the communication cost to a worst case $O(logN)$ too.

Assumption 2.4 (Routing cost) *Provided the information state from Assumption 2.3, the network diameter for a SPN is $O(\log N)$. That is, in the worst case, the number of visited nodes between any two nodes is logarithmic in respect of the network size.*

The information state of any node p consists mainly in a routing table of other nodes, to wit *neighbors*. Given a message *msg*, the node p deterministically establishes the neighbor t to who send the message *msg*, so that the node q responsible for message *msg* is reached in the aforementioned routing cost. To do so, most of the SPNs utilize a *greedy* routing algorithm.

Definition 2.1 (Greedy routing) *A greedy routing in graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where vertexes $\mathcal{V}$ are the nodes, and edges $\mathcal{E}$ the links between nodes, with distance function $\omega : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}^+$ entails the following decision: Given the target node q responsible for message msg, node p with neighbors $\mathcal{V}_p = getLinks(p)$ forwards msg to node $n \in \mathcal{V}_p$ such that $\omega(p,q) = min_{n \in \mathcal{V}_p} \quad \omega(n,q)$.*

That is, any node p consulting its local information state can discern which is the best choice to forward a given message *msg*. In other words, complementary to the above Definition 2.1, we can state the following remark.

Remark 2.2 (Keyspace addressing) *Let $segment(p,n)$ be the $\mathcal{I}$'s segment that node n is responsible for, using only the local information state of node p. Namely, if a message msg must be forwarded to a key $k \in segment(p,n)$, node p will draw n to forward msg. Let $\mathcal{V}_p = getLinks(p)$ be the p's neighbors. The local state information of any node p in an SPN is organized so that $\mathcal{I} = \bigcup_{n \in \mathcal{V}_p} segment(p,n)$.*

Consequently, the SPN routing algorithm provides the necessary decision criteria to draw a node's neighbor to forward any given message, so that the network diameter is guaranteed. In addition, most of the SPNs provide *path convergence* which is defined as follows:

Definition 2.2 (Path convergence) *A SPN routing scheme provides path convergence whenever paths to a common destination intersect at a rate proportional to the distance between the source nodes.*

2.1.2 Hierarchical Structured Peer-to-Peer Networks

Most of the SPNs are flat networked structures, so that all nodes are considered to be in the same logical plane, where all nodes share the same responsibilities. There are other SPNs structures called *hierarchical*, though. A *hierarchical* SPN reckons that nodes appear at different (logical) layers, building several sets of nodes, namely *clusters*, where the set of clusters establish a tree-like network structure which gathers all participating nodes. Marc Sanchez *et al.* [44] introduced the different models of constructing hierarchical peer-to-peer networks: super-peer-based model (like [45]) or a homogeneous design (like [42]).

A *super-peer-based hierarchical model* has the ability to join (potentially different) flat SPNs into a single system. To do so, these systems establish a set of nodes as *super-peers*. The role of a super-peer is to interact with any flat SPN it lives as a normal node, but also as a connector to other clusters of nodes through other super-peers. The idea behind the super-peer-based model is to leverage the node heterogeneity, so that the most powerful nodes can be elected as super-peers. This way, super-peers would be able to process all the intra-cluster and (potential) inter-cluster traffic that they must address. A super-peer model usually builds a two-tier hierarchy, but in some designs the hierarchy's height can be self-adjusted according to the number of participants (like in ZigZag [46]). This way, the super-peers of the leaf clusters can be selected again to play the role of super-super-peers for the immediate higher level, and so on. Nevertheless, it is clear that the main drawback of this design is the super-peer selection algorithm, since the system's performance can degrade notably if wrong candidates (e.g., nodes with little process capacity or a little bandwidth) are chosen as super-peers.

Conversely, the main characteristic of *homogeneous designs* is that all peers are considered as equal. Assumptions on equality are made regarding peers characteristics, such as available bandwidth and storage capacity. As a consequence of this assumption, homogeneous designs can exploit the uniformity of load and functionality that characterizes flat SPNs, such as Chord [7], Pastry [8] and Kademlia [15], but they enhance the whole solution by incorporating new features like fault isolation and path locality. To illustrate, we elaborate on Cyclone [42], a homogeneous hierarchical design. Its matchless properties, such us *genericity* and flexible *clusterization mechanism*

to say the least, motivate its overall description in the following and have propelled to be considered as a substrate in some of our services. Briefly speaking, Cyclone is *generic* from the SPN viewpoint, since Cyclone is a framework targeted to construct hierarchical versions of flat SPNs. In addition, the Cyclone's *clusterization mechanism* is flexible. That is, Cyclone uses part of the node ID to group nodes into clusters (and sub-clusters), which is very attractive for particular self-organization of clusters according to a certain criteria (like geographical proximity).

2.1.2.1 Cyclone

Cyclone [42] features the homogeneous hierarchical design by using a single keyspace for all nodes in the system. In Cyclone, each node is contained in a collection of telescoping clusters, rather than in a single separate cluster. The collection of clusters is defined so that any two clusters are either disjoint or one is a proper subset of the other. The tree-like structure if formed by a set of disjoint leaf clusters, that are joined consecutively at higher levels, where the root is the top cluster.

Actually, Cyclone provides a framework to construct hierarchical versions of flat SPNs, which we call *instantiated* SPNs. For instance, Whirl [42] instances the hierarchical design of Chord. The main Cyclone's characteristics and properties are detailed in the following. We assume flat SPNs with logarithmic node degree and network diameter $O(\log N)$ such as Chord [7], where N is the number of nodes.

Node identifiers and keys. Having a m-bit keyspace (i.e., $[0..2^m)$), the cluster identifier *clusterId* is formed by a suffix of cl bits, $cl < m$, and $cl \geq l$, where l is the cluster level (illustrated in the Fig. 2.1). In other words, the cl bits correspond to the less significant bits (LSB) of a key.

The node identifier *nodeId* for a node at l'th hierarchy level is formed by the prefix of $m - cl$ bits, namely the $m - cl$ most significant bits (MSB). This *nodeId* is used within the intra-cluster overlay protocol (e.g., Chord in the case of Whirl).

Remark 2.3 (Cyclone partition of the ID) *The ID of a node in Cyclone is divided into two parts, the clusterId and the nodeId. The former identifies the cluster provenance, and the nodeId is used by the instantiated SPN to route within the cluster.*

Information state. The routing table of a node p participating in Cyclone is slightly modified, guaranteeing at any time a logarithmic degree though. To do so, the routing table is filled in according to the algorithm of the instantiated SPN, starting from leaf

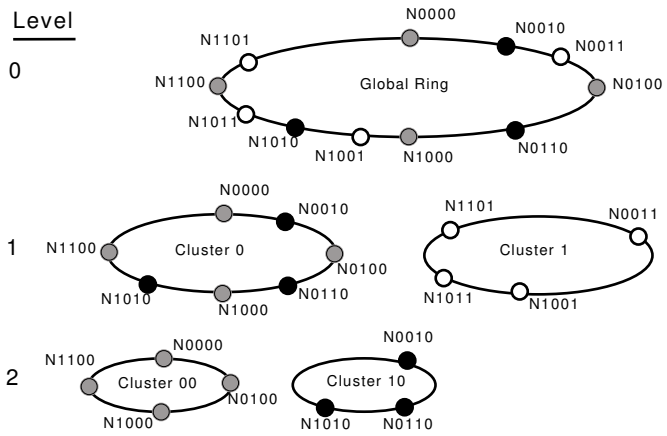


Figure 2.1: Example of a Cyclone setting, describing the location of nodes within the clusters. The instantiated SPN is a Chord-like network.

cluster C and following up to higher level clusters, until the root cluster is reached. Nevertheless, this routing table only allows a *conventional routing* as in the instantiated SPN. It does not permit to route against a sibling cluster.

To this end, Cyclone constructs a particular routing table, namely *XOR routing table*, which allows the communication between sibling clusters in a logarithmic cost. The degree of this routing table is a small $O(\log |\mathcal{C}|)$, where $|\mathcal{C}|$ is the number of clusters. The table is filled in with nodes that suit the XOR metric, like in Kademlia [15]. The difference from Kademlia is that Cyclone only computes the XOR metric into the $clusterId$. In conclusion, the total degree for a node participating in Cyclone is $O(\log N + \log |\mathcal{C}|)$, corresponding to the *conventional* and *XOR* routing tables, respectively.

Remark 2.4 (Cyclone's information state) *Any node in an instantiated SPN maintain the conventional and the XOR routing tables.*

Routing algorithm. There are two routing schemes in Cyclone: *conventional* and *XOR routing* algorithms. The conventional routing obeys to that of the instantiated SPN, but in a bottom-up fashion. A node p always sends a message m with target key k into the p's leaf cluster C, and m is routed as if only that cluster exists. The message m arrives at k's responsible node in C, which is called the k's exit point in C, namely EP_C^k. At this moment, EP_C^k will continue the routing process in the immediate higher cluster C', forwarding m through C' until k's exit point $EP_{C'}^k$ is reached. The routing process is repeated consecutively, concluding when the absolute responsible node for

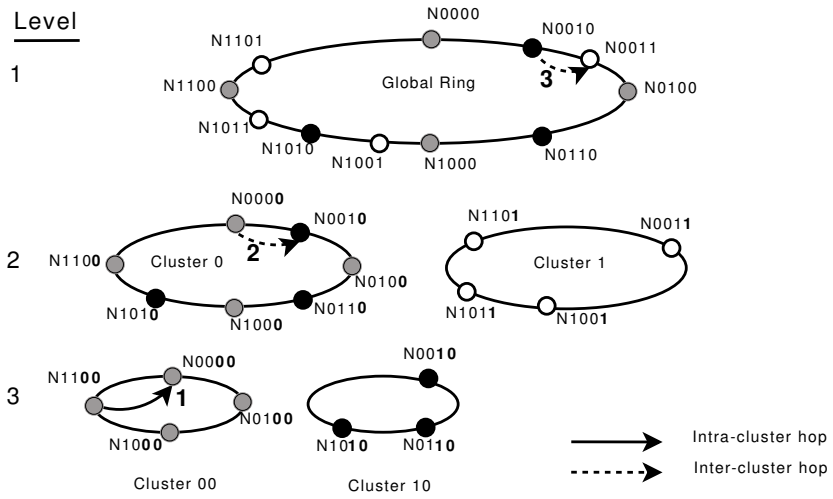


Figure 2.2: Example of (bottom-up) *conventional routing* in Cyclone. Node *N1100* sends a message with key *K0011*. On every cluster, the SPN's conventional routing is applied, discarding the *clusterId* bits accordingly at each level. The exit points are *N0000* for cluster *00* (EP_{00}^{K0011}), *N0010* for cluster *0* (EP_{0}^{K0011}) and *N0011* for global cluster (EP^{K0011}), which finally corresponds to the responsible node for the key.

k (i.e., $EP_{C^R}^{k}$) is reached at the root cluster C^R. An example is shown at Fig. 2.2. The communication cost is bounded to $O(\log N)$.

Remark 2.5 (Cyclone's exit point nodes) *At any given Cyclone's cluster C exist a node p that is responsible for the key k, which is called the exit point node for k, namely* EP_{C}^{k}.

Complementary, the *XOR routing* uses the *XOR routing table* and allows the navigation through the hierarchy of clusters, iterating against sibling clusters. To do so, Cyclone routes greedily to the destination cluster C', selecting the XOR entry that minimizes the distance to the target key k. The routing reaches C' when the bits of the k's $clusterId$ equals to the bits of a p's $clusterId$ from a node p, which determines that p is a node living in C'. After reaching the destination cluster C', the *conventional* routing applies until the k's responsible node $EP_{C'}^{k}$ is found, which concludes the routing process. We illustrate this process in Fig. 2.3. The communication cost is yield to Δ, where Δ is the number of bits for the binary representation of $scId \times tcId$, being $scId$ and $tcId$ the p's and k's $clusterIds$, and $\times$ the bitwise exclusive OR. The XOR routing is bounded then to $O(\log |\mathcal{C}|)$.

Remark 2.6 (Cyclone's routing algorithms) *The conventional SPN routing provides a bottom-up routing algorithm, whilst the XOR routing algorithm provides a top-down routing as well as routing to sibling clusters.*

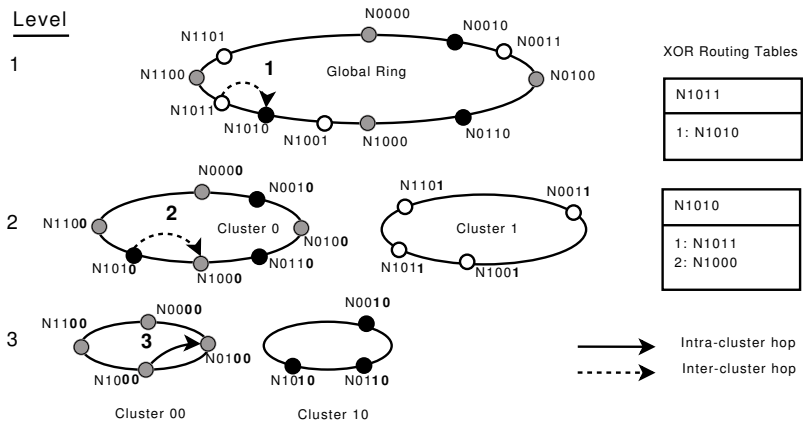


Figure 2.3: Example of (top-down) *XOR routing* in Cyclone. Node *N1011* needs to locate the cluster codified as *00* where to look for a key *01*. Steps 1 and 2 are part of (top-down) XOR routing in order to locate the destination cluster. Step 3 is the part of *conventional routing* to locate the node responsible for the key *K0100* into the cluster *00*, namely the EP_{00}^{K0100}.

Node organization and locality. Cyclone permits a flexible organization of nodes into the hierarchy. Let $\mathcal{M}_{\mathcal{P}} : \mathcal{P} \times \mathcal{P} \longrightarrow \mathbb{R}$ be a metric function, namely metric, that measures the *proximity* between any two nodes with regard to an arbitrary node's property. Therefore, nodes could join the system and be distributed according to metric $\mathcal{M}_{\mathcal{P}}$ (such as physical-network or geographical proximity): Leaf clusters would gather near nodes (with regard to $\mathcal{M}_{\mathcal{P}}$), and higher level clusters would collect farther and farther nodes, respectively.

Cyclone's routing algorithm takes advantage of the proximity metric, firstly looking up in leaf clusters and then, if necessary, forwarding greedily the query to a foreign cluster, via the exit points, until the responsible node is reached. Irrespective of the instantiated SPN, this contributes in new features like fault isolation and path locality.

Remark 2.7 (Cyclone's node organization) *The distribution of nodes in Cyclone can be adapted to flexibly organize nodes into clusters according to an arbitrary metric $\mathcal{M}_{\mathcal{P}}$.*

Load balancing. Cyclone selects exit point nodes for data caching and replication, achieving an efficient load balancing. Intuitively, the first lookup will retrieve the data object from the (cluster-foreign) owner node, but extra lookups will match on local cluster's exit point nodes. This confers to Cyclone a natural ability for data load balancing under repetitive data operations (like searches) and reduction of the completion time on successive data operations.

Remark 2.8 (Cyclone's load balancing) *Exit point nodes at Cyclone can contribute on augmenting transparently the system load balancing.*

2.1.3 Peer-to-Peer Systems: Summary

In this section we have presented the existing sorts of peer-to-peer systems. We have motivated that *structured* peer-to-peer systems (SPNs) perform better than the counterpart *unstructured* ones (UPNs), since UPNs do not provide all necessary properties to set up high-level functionality (where *query correctness* embodies the set of expected system properties). This has propelled the introduction of the common properties of SPNs to be considered in the rest of this document. As a calculated side-effect, we have also shown the nomenclature that will be used along this thesis.

The last part of this section details the characteristics of hierarchical peer-to-peer systems, differing from the *super-peer-based* and the *homogeneous* model. In particular, we have described Cyclone [42], a homogeneous design of a hierarchical peer-to-peer system. The motivation behind that is because its good properties made Cyclone suitable as the substrate in some of our services.

In the following sections, we perform an analysis on the state of the art for both similarity queries and, afterwards, publish/subscribe systems. The last concluding remarks will close this chapter.

2.2 Data Management Services

The data management services we discuss in this chapter belong to the family of similarity queries. To provide an outline on the kind of systems we consider, we introduce in the following section a couple of scenarios where similarity queries are necessary. Afterwards, we state in Section 2.2.2 an overview of a generic design of a system providing similarity queries (See Fig. 2.5). In addition, for the lack of clarity, we provide in Section 2.2.3 a formal definition of the sort of similarity queries we consider: range queries, k-NN queries and spatial queries.

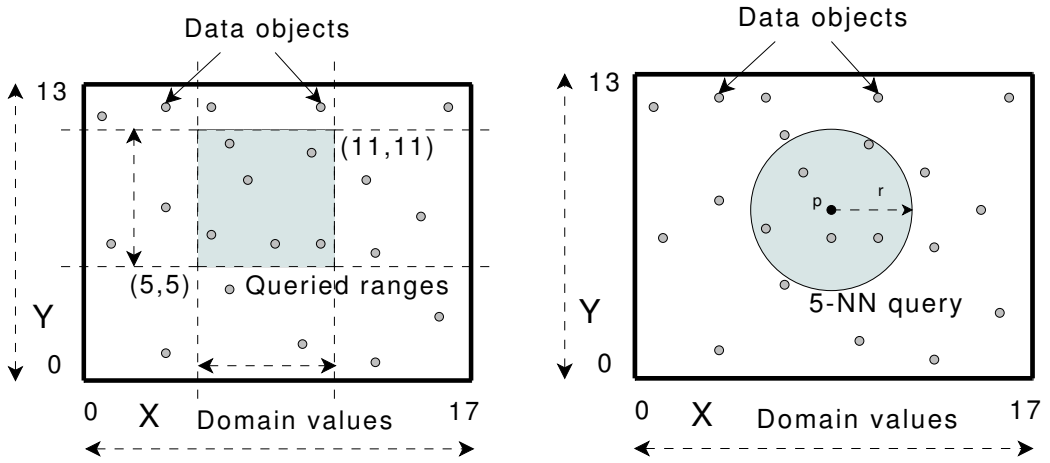
2.2.1 Use Cases: Similarity Query Applications

To let the reader understand the necessity and complexity of providing similarity queries over distributed systems based on SPNs, we explain two applications where such a functionality is required, among scalability by service decentralization and high performance by operation parallelization. Current real-life applications cannot be understood without such richer abstractions.

The first application we consider is the **document retrieval** in a distributed network of workstations (namely nodes). *Documents* consist of any kind of text documents and the goal of the application is to provide similar documents according to the *user interests*. User interests are specified by means of keywords, like "Mathematics" or "Computer Science". Actually, each search is an instance of a **k-nearest neighbor query** (k-NN): the system retrieves the *k most similar documents* to the user interests (See Fig. 2.4b for an example). The key procedure of this kind of applications is to treat each document as a pair of the form $\{< keyword - list >, < document >\}$. The *keyword-list* is a list of keywords that contains the most representative descriptive terms from the *document*. This technique is called *Information Retrieval* (IR), for what several ways of parallelizing document indexation and search have been widely studied [24]. Particularly, we focus on the distributed document retrieval, so that different networked nodes must be asked for retrieving related documents to the user query.

The second application we consider is a distributed **geographical service location**, so that system participants are networked nodes that possess partial information and help on query resolution. For simplicity and for the proposal of the analysis, we consider the goal of this application is telling the *services location* related to the *user interests* that appear in an arbitrary *geographical area*. This exemplifies the behavior of a **range query**: retrieve all elements from the given range, that is, the geographical area. In addition, this is also an example of **spatial query**, because the goal of the application is locating information from a geographical region (See Fig. 2.4a for an example). This case, *user interests* determine both the geographical area and the sort of services she is looking for. For instance, the reader can be interested in retrieving the "Italian Restaurants" locations around her city. The idea of this application is to retrieve the whole set of items that appear in a rectangular area of arbitrary size. This is however a hot topic on the study of Geographical Information Systems, where the feasibility of the distributed resolution of the search has largely been studied for different environments (for instance for Grid computing [47]). Clearly, distributed and parallelized queries accelerate search operations, reducing the response time, but at the cost of a higher use of communication bandwidth.

All these (distributed) applications present the same elements in their architecture: a *multi-dimensional data domain* with data objects being stored and searched; a *distance function* which defines the similarity between two data objects; and the *algorithms* for storing and searching the content. For instance, IR applications characterize each text document by a d-dimensional vector with the best descriptive terms. Geographical service location applications consider each service generically as being a



(a) Range query example. **(b)** k-NN query example.

Figure 2.4: Examples of similarity queries on a 2-dimensional scenario. In both cases, the shadowed area represents the *covered* area for the given query. In (a), the range query covers the area $q = \{[5..11], [5..11]\}$, whilst the k-NN in (b) asks for 5 objects that lie within the query $q = \{p, r\}$.

d-dimensional vector, including the latitude, longitude and meta-data from the service. In both cases, a search is processed by evaluating the similarity of various data objects according to some distance function, e.g. Cosine distance for IR ones and Euclidean distance for service location ones. Out of these three elements, we include into the study both the multi-dimensional data domain and the algorithms. Distance functions are not considered because they are tightly related to the application context, with no effect on the distributed system.

2.2.2 System Design: An Overview

Having observed the above use cases, now it is turn to complete the view of systems that aim at providing similarity queries. We provide an overview of a generic implementation structure of such applications. As we can see from Fig. 2.5, the system design is composed by (several or thousands of) individual instances. Each instance (or node) is constituted by the *Node* and *Application* layers, bestowing specific functionalities on either. The *Node* layer guarantees the communication within the distributed system by means of a routing table where some *neighbors* appear. A message is sent to a neighbor according to some of the available *routing algorithms*. A *Node* can manage different routing procedures each of which is related to a certain kind of complex operation, such as range or k-NN query. Lastly, a *Node indexes the data* being stored in order to accelerate the resolution of the enhanced queries. Upon the reception of an

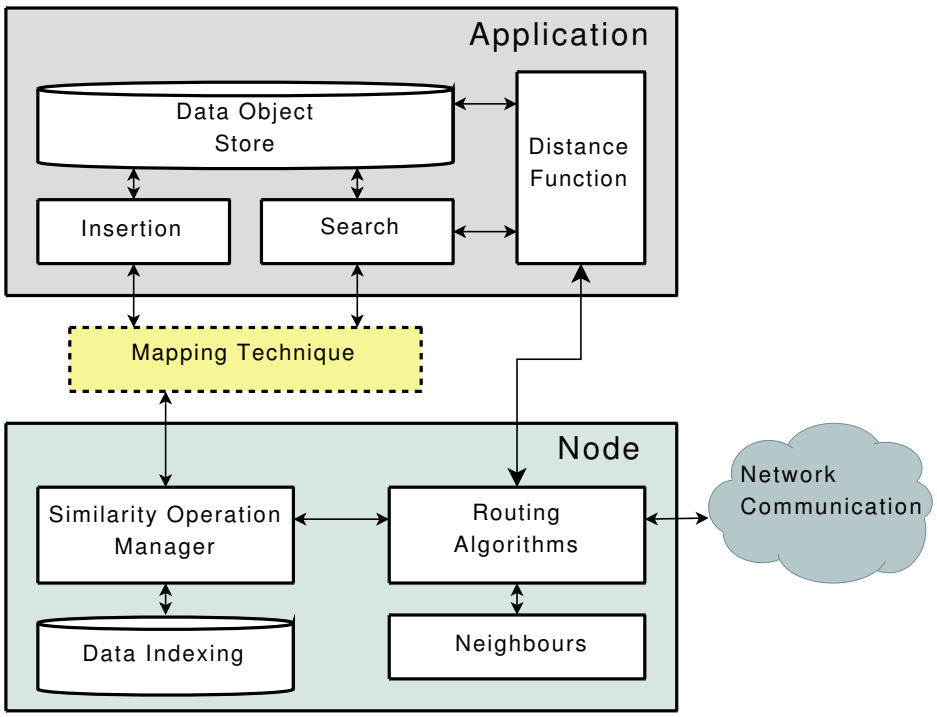


Figure 2.5: Generic layered design, common components and information flow of distributed systems providing similarity abstractions.

operation message, its *operation manager* is drawn to help in the resolution. One of its tasks is to decide whether the operation ends in the current instance or it must be forwarded to other neighbors. The operation is concluded by accessing to some building blocks in the *Application* layer.

The data indexation can be performed as is, which means that no adaptation is applied to the data object, or the system can use a *mapping technique*, namely $\mathcal{F}_{\mathcal{O}}$, which adapts the object to the keyspace of the underlying SPN.

The *Application* layer provides local *data management* services, such as *insertion* and *search* operations. The search services employ the *distance function* to compute the proximity among data objects. Eventually, the *Application* is in charge to *store* the set of data objects which a node is responsible for.

To illustrate, and recovering one of our examples, IR applications embody k-NN query resolver systems: the application retrieves an arbitrary amount of similar documents to the user-selected keywords. The number of elements in the result set is defined to be as maximum as a system-wide or user-modifiable property value. The system calculates the distances from the user-selected keywords to the d-dimensional descriptive terms of documents and builds a result set with the k closest documents.

The idea is similar to have all documents ordered in a list from closer to farther to the user-selected words and selecting the first k documents from the list. A centralized solution takes advantage of disposing locally all information, possessing a global knowledge of the dataset, and then employing efficiently IR techniques to produce the result set. The problem appears when the system is widely distributed among nodes for scalability, as well as to join the efforts and resources of users. The way that documents and information are distributed through nodes will drastically determine the system's efficiency and response time.

2.2.3 Similarity Queries: Definitions

Broadly speaking, the **similarity queries** family provides a set of abstractions to (distributed) applications in order to discover alike, *similar* objects to user interests. Let $\mathcal{M}_{\mathcal{Q}} : \mathcal{Q} \times \mathcal{O} \longrightarrow \mathcal{D}$ be the proximity metric from data items to the given query, where $\mathcal{Q}$ is the query domain, $\mathcal{O}$ the data items domain, and $\mathcal{D}$ the distance domain.

Definition 2.3 (Similarity query) *The result of a given similarity query q is formed by the set $\mathcal{A} = \{o \in \mathcal{O} | \mathcal{M}_{\mathcal{Q}}(q, o) < \delta\}$, for a certain value δ.*

A similarity query can be seen also as the set of objects contained in the interior of a hyper-sphere, with center q and radius δ. The number of elements in the result set will depend on δ and the similarity algorithm itself.

The **range queries**, also known as *window queries*, are a subset of the similarity queries. In particular, range queries are to provide the whole set of elements into the specified range. There are two approximations to it. Let us call them as **sphere-based** and **region-based** approaches. In a **sphere-based approach**, the range query is defined as a pair $q = \{p, r\}$, where $p \in \mathcal{O}$ is a point in data domain, and r is called the radius.

Definition 2.4 (Sphere-based range query) *The result set of a sphere-based range query is formed by the objects in $\mathcal{A} \subseteq \mathcal{O}$, $\mathcal{A} = \{o \in \mathcal{O} | \mathcal{M}_{\mathcal{Q}}(p, o) < r\}$.*

It is easy to see that the set of objects that belong to $\mathcal{A}$ are objects that live in the interior of a hyper-sphere with center p and radius r. On the contrary, in the **region-based approach**, the range query for a d-dimensional data domain $\mathcal{O}$ is considered to be $q = \{[l_1, h_1], [l_2, h_2], \ldots [l_d, h_d]\}$ (i.e., a hyper-region in a hypercube), where every l_i and h_i, $1 \leq i \leq d$, are the lower bound and the higher bound for the i-th dimension, respectively.

Definition 2.5 (Region-based range query) *The result of a region-based range query is defined to be the set* $\mathcal{A} \subseteq \mathcal{O}$, $\mathcal{A} = \{o \in \mathcal{O} | \forall i \in [1, d] : l_i \leq o_i \leq h_i\}$, *where* o_i *refers to the value of object* o *at the* i*-th dimension.*

The **k-NN queries** are a subset of the similarity queries that aim at providing the k nearest objects to a given query $q = \{p, \delta^*\}$, where $p \in \mathcal{O}$ is a point in the data domain, and δ^* is an adjustable radius.

Definition 2.6 (k-NN query) *A k-NN query asks for the set of objects* $\mathcal{A} \subseteq \mathcal{O}$, $\mathcal{A} = \{o \in \mathcal{O} | \mathcal{M}_Q(p, o) \leq \delta^*\}$, *such that* $|\mathcal{A}| \leq k$, *where* $|\mathcal{A}|$ *accounts the number of elements in the set* $\mathcal{A}$, p *is the center of the hyper-sphere, and* k *the number of elements to retrieve.*

This case, the radius δ^* is progressively adjusted so as to leave $\mathcal{A}$ with just k elements. Usually δ^* takes a little value so all k items are not retrieved at once, but after some few repetitions of the k-NN query while incrementing the value of δ^* until $|\mathcal{A}| = k$.

To conclude, **spatial queries** are a particular case of either range or k-NN queries, depending on the application context, where one of the dimensions to be considered is *the location of the data objects*.

2.3 Parallel Computing on Similarity Queries

Similarity searches onto SPNs benefit from scalability by service decentralization and high performance by query parallelization. Since the service decentralization is clear because they are set up onto a SPN, we delve into query parallelization in this section. Parallel computing of queries provides high performance to the system and this depends mainly on the way nodes are interconnected. For an analysis on the factors that enable parallel computing in a distributed setting refer to Section 2.4.3.

The way that nodes are organized into the SPN is known as **topology**. Topology influences the communication costs between participants and the way that information is stored into nodes for global efficiency. The most common peer organizations among SPNs are flat topologies (e.g., rings [7, 26], torus [11]) and hierarchical topologies (e.g., trees [44]).

As we have detailed before, similarity queries considered into this chapter are broadly of two types, so that we describe two generic algorithms, one for range and another one for k-NN queries. We do not delve into spatial queries because they can be a sub-set of any of them, according to the system's approach. In the worst case,

queries are initiated by nodes (namely *querying nodes*) that are not responsible for resolving the query. This turns our algorithms to broadly describe a two-phase problem solver. The first phase consists of routing the query until nodes responsible to answer the query are reached. Afterwards, the algorithm collects all data objects that belong to the result set during the second phase. These algorithms provide a uniform and generic approach to solve the queries. Thus, the reader will realize that systems considered into this study employ adapted algorithms to suit the particular systems' architectures.

2.3.1 Range Query Parallelization

Algorithm 2.1 *range_query* gives the details of how range queries are resolved. Their resolution is fully parallelizable (lines 3–6), so that nodes provide to the querying node their (partial) result sets (line 9). This algorithm employs several local functions. Function $subqueries(node)$ returns the (keyspace) segments that *node* is responsible for. This way, *lquery* (line 1) takes a part of the *query* (namely sub-query) that *node* can answer, what leaves in *oquery* (line 2) the part from the *query* that must be resolved by other nodes. Function $subquery \in oquery$ (line 3) provides all different *subqueries* from *oquery*, where every *subquery* is a sub-query that local *node* is able to start in parallel. Whenever $oquery = \emptyset$, this function provides no *subquery* so that the loop is never executed. Function $best\_neighbor(node, subquery)$ returns the best *node*'s candidate neighbor from *node*'s routing table to which forward the *subquery*. This selection is SPN-specific and is dictated by system's routing scheme. Function $data(node)$ provides the data that *node* possesses (line 7). Once a *node* has a (partial) $result\_set$, it sends back to the querying node *qnode* by means of the function $send(node, qnode, result\_set)$ (line 9).

As it is explained before, (sub-)queries are only initially routed (lines 3–6). Once responsible nodes are reached, lines 3–6 distributes the (sub-)queries until the whole query is processed ($query = \emptyset$). Most of hierarchical and some of the flat topologies provide *disjoint* paths (sequences of nodes) to reach other nodes, what makes Algorithm 2.1 very effective. Thus, in this scenario, this algorithm parallelizes both routing and query resolution. But most of the flat topologies usually provide *path convergence* (see Definition 2.2), as well as *data locality* preservation (so that neighboring nodes own close data objects). In practice, path convergence and data locality imply that the different (sub-)queries sent by the querying node travel the network along the same path. Therefore, these messages waste bandwidth and computer resources until the first responsible node of the query is reached. This is why topologies with path convergence

Algorithm 2.1 *range_query*
Input: *node* /* *node where algorithm is executed* */
Input: *query* /* *range query* */
Input: *qnode* /* *querying node* */

1: $lquery \leftarrow query \setminus subqueries(node)$
2: $oquery \leftarrow query \setminus lquery$
3: **for all** $subquery \in oquery$ **do** /* in parallel */
4: $neigh \leftarrow best\_neighbor(node, subquery)$
5: $range\_query(neigh, subquery, qnode)$
6: **end for**
7: $result\_set \leftarrow data(node) \bigcap lquery$
8: **if** $result\_set \neq \emptyset$ **then**
9: $send(node, qnode, result\_set)$
10: **end if**

and preserving data locality improve the *range_query* algorithm (see Algorithm 2.2) by avoiding query splitting whenever query is only routed ($lquery = \emptyset$ in line 2).

2.3.2 k-NN Query Parallelization

Algorithm 2.3 provides a big picture on the way k-NN queries are worked out. For the sake of simplicity, this algorithm is based on a naive linear scan, even though other approaches (e.g., space partitioning, locality sensitive hashing and other approximate nearest neighbor search) are actually used by systems in order to improve k-NN query performance.

The *knn_query* algorithm is as follows. Firstly, a querying node *qnode* performs the k-NN query by specifying a pair $q = \{p, r\}$ and k, where p is a point or data object, r defines an initial radius and k the number of data objects to fetch. This operation initially consists of a single message routing (lines 1-3), which cannot be parallelized. When the *corresponding node* responsible of p is reached (line 4), it initiates an *expanding ring* search (lines 5-18). This search describes a linear scan through close nodes to corresponding node, in order to provide at most k objects that are not farther than r from p. This algorithm though takes the assumption that closer nodes will always provide closer data objects to p than farther nodes. Thus, the linear scan is consecutively *expanded* in parallel to farther nodes from corresponding node, since the k-NN query is not resolved (line 9). When no new data objects are appended to the candidate result set (having $result\_set = new\_result$ in line 9), the expanding ring search stops and the k-NN query concludes (line 19).

Algorithm 2.2 *range_query* with path convergence
Input: *node* /* *node where algorithm is executed* */
Input: *query* /* *range query* */
Input: *qnode* /* *querying node* */

```
 1: lquery ← query \ subqueries(node)
 2: if lquery = ∅ then
 3:    neigh ← best_neighbor(node, query)
 4:    range_query(neigh, query, qnode)
 5: else
 6:    oquery ← query \ lquery
 7:    for all subquery ∈ oquery do /* in parallel */
 8:       neigh ← best_neighbor(node, subquery)
 9:       range_query(neigh, segment, qnode)
10:    end for
11:    result_set ← data(node) ⋂ lquery
12:    if result_set ≠ ∅ then
13:       send(node, qnode, result_set)
14:    end if
15: end if
```

As before, this algorithm employs some *node*'s local functions. The condition '*node* is not responsible for p' is only $false$ whenever *node* owns p. This means that if p existed, *node* would be the node to hold p. Function $best\_neighbor(node, p)$ (line 2) is the same as before, but where the neighbor is selected according to the searching point p instead. Function $knn(p, k, r, \{dataset\})$ (lines 7, 12, 16) provides at most k data objects from *dataset* that are not farther than r from p. At line 7, r has not been setup, so that a worst case ∞ value is given. Function $expanding\_ring(node, visited\_nodes)$ (line 11) returns a list of nodes that should be visited in the next query's round as the expanding ring search dictates. Note that *visited_nodes* holds the set of nodes close to *node* that the expanding ring scan has already visited. Line 12 tells *node* to wait for the results *partial_result* from neighbor node *nnode*. Thus, an inter-node communication is performed here between *node* and *nnode*. Function $send(node, qnode, result\_set)$ (line 19) is the same as before, even though this function is only executed once in *knn_query* algorithm.

These algorithms provide a global view on how the aforementioned similarity queries work. We provide in the following section the significant parameters with which we will evaluate all systems considered into this study.

Algorithm 2.3 knn_query
Input: *node /* node where algorithm is executed */*
Input: *p /* point of the k-NN query */*
Input: *k /* number of elements for the k-NN query*/*
Input: *qnode /* querying node */*

1: **if** *node* is not responsible for p **then**
2: $neigh \leftarrow best\_neighbor(node, p)$
3: $knn\_query(neigh, p, k, qnode)$
4: **else**
5: $visited\_nodes \leftarrow \emptyset$
6: $result\_set \leftarrow \emptyset$
7: $new\_result \leftarrow knn(p, k, \infty, data(node))$
8: $r \leftarrow longest\_distance(p, new\_result)$
9: **while** $result\_set \neq new\_result$ **do**
10: $result\_set \leftarrow new\_result$
11: **for all** $nnode \in expanding\_ring(node, visited\_nodes)$ **do** /* in parallel */
12: ask *nnode* for $partial\_result \leftarrow knn(p, k, r, data(nnode))$
13: $new\_result \leftarrow new\_result \bigcup partial\_result$
14: $visited\_nodes \leftarrow visited\_nodes \bigcup \{nnode\}$
15: **end for**
16: $new\_result \leftarrow knn(p, k, r, new\_result)$
17: $r \leftarrow longest\_distance(p, new\_result)$
18: **end while**
19: $send(node, qnode, result\_set)$
20: **end if**

2.4 Evaluation Criteria

The systems providing similarity queries target of this study are analysed accordant with a set of qualitative and quantitative parameters, constituting a *common evaluation framework* (see Table 2.1). For a fair comparison, it is highly recommendable to consider both quantitative and qualitative parameters that describe the key systems' components. In addition, such an evaluation framework provide the systems' portrayal and facilitates the comparison between systems. The considered parameters refer to structural, efficiency, as well as behavioral properties of systems. The parameters can be considered broadly of two types: a) **implementation** parameters, that refer to the system design and construction, and b) **quality of service** parameters, that consider the quality of the query resolution. An analysis of parallel computing abilities according to these properties is also included.

2.4.1 Implementation Criteria

Which parameters do we consider for their inclusion into the evaluation framework? Let us consider the geographical service application to this end, distributed over the world onto an arbitrary number of nodes. As we have seen before, the way that nodes are organized (namely the **topology**) dictates the communication costs between nodes and the way that information is stored into nodes for global efficiency. This motivates the inclusion of the topology into the evaluation framework.

Table 2.1: Characterization of the evaluation criteria for systems providing similarity queries.

Criteria	Characterization
Mapping properties	
Topology	The **topology** defines a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where vertexes $\mathcal{V}$ are nodes, and edges $\mathcal{E}$ the links established between nodes. In addition, only edges $\mathcal{E}$ are used for node inter-communication in normal peer-to-peer operations. The topology can be broadly specified as *structured*, when the topology relies on any geometric form (e.g., ring, hypercube, tree), and *unstructured*, when nodes are connected in a non predefined fashion (e.g., a mesh).
Dimensionality	Systems that provide this kind of high-level services in a distributed way can support either one-dimensional (**1-D**) or multi-dimensional (**M-D**) application domains. When possible, systems will be analysed for both sorts of dimensionalities.
Mapping approach	Because in most cases application domains do not correspond to SPN keyspaces, the distributed application needs to adapt the information in order to allow its indexation by the SPN. This kind of data transformation is called also *mapping* and defines exactly the way the transformation is achieved. We can find these sorts of transformation: **1:1** (Left) One application dimension is mapped to a single SPN keyspace (right). **M:1** The whole multi-dimensional application domain is mapped to a value in the SPN keyspace. **M:P** The whole multi-dimensional application domain is transformed to a P-dimensional SPN keyspace. **N/M** When no mapping is applied.

Continued on next page

Table 2.1 – *Continued*

Criteria	Characterization
Insertion and search properties	
Storage efficiency	In a distributed or parallel system, the amount of storage used to record information is significant to reduce notably the response time. In general, the more copies of the same information, the less delay on answers. We use this property to quantify the *overhead* that different algorithms pose on storage.
Time efficiency	This property measures the amount of time needed by the distributed algorithm to return the whole set of results. Because in most of the algorithms nodes with requested information answer directly to querying node, this time is considered to be equivalent to the amount of time the algorithm takes for visiting the last node involved in the query. When noted, it also means the time that a first (partial) result can be provided from the system.
Load balancing	When operating in a distributed or parallel system, it is convenient that all nodes have (approximately) the same amount of *load*. The term *load balancing* here means both *data* and *routing load balancing*: the former property tells whether all nodes (approximately) manage the same amount of information; the latter depicts that the system is able to route throughout the set of nodes without hotspots (i.e., nodes with high routing process).

Continued on next page

Table 2.1 – *Continued*

Criteria	Characterization
Result set properties	
Correctness	When all elements o from the result $\mathcal{O}'$ of a given query q are all in the set $\mathcal{A}$ of possible answers for q (i.e., $\mathcal{O}' \subseteq \mathcal{A}$), is said that the result $\mathcal{O}'$ is correct and, by extension, the distributed algorithm used to resolve the query. In other words, $\{\forall o \in \mathcal{O}' \| o \notin \mathcal{A}\} = \emptyset$. This property also means that if the system has some possible result for the given query q, it is found.
Completeness	This property is somewhat complementary to the property above, in such a way that *correctness* does not ensure that *all* elements matching the query are returned in the result set. Thus, a distributed algorithm is said to provide *completeness* in its results if and only if $\mathcal{O}' \supseteq \mathcal{A}$. Note that $\mathcal{O}'$ can contain some elements that actually are not part of the expected answer $\mathcal{A}$.
Intersection	On any kind of query, results provided to the querying node can appear already selected with only those items that really are interesting for the user. In this case, the *intersection* of the found data with the query's predicate has been done on the *system side*. Otherwise, results needs a last step in the *user side* to prune all items that actually are not useful to the user.

We know how nodes are interconnected, but how are nodes addressed into the system? As we have introduced in Section 2.1.1, every SPN specifies a *keyspace* $\mathcal{I}$ from where node identifiers (namely *ID*s) are drawn. This is done by the use of a function $\mathcal{F}_{\mathcal{P}}$, which constructs an ID $\in \mathcal{I}$. Most of the SPNs use a unidimensional keyspace (like in Chord [7]), but there exist another family which uses a *multi*-dimensional keyspace (like in CAN [11]), where nodes are identified by a vector of values of the form $\{id_1, id_2, ..., id_M\}$. However, application data domains (e.g., geographical location of services) do not resemble SPN keyspaces (e.g., Chord keyspace) with respect to **dimensionality**, type of information and/or data domain.

Therefore, SPNs require **mapping techniques** $\mathcal{F}_{\mathcal{O}}$ to support data object indexing, that convert and adjust application data domains to the SPN keyspace. In addition, this mapping must be deterministic and common for all nodes, so that for the same data object any node will produce the same key. Examples of mapping techniques are hash functions like SHA-1, space filling curves (SFCs) like Z-curve [40] or Hilbert Curves [35], or based on the iDistance [48]. Traditionally SPNs make use of uniform hash functions to distribute information uniformly at random between nodes for load balancing purposes. Their drawback is the *loss of data locality*. That is, two near data

objects in the application domain will be randomly placed into the system, so that close nodes will not store similar data objects. Similarity queries cannot be efficiently deployed in this scenario. The naive solution, thus, would be broadcasting the similarity query to all nodes. Of course, this approach does not scale for large distributed systems and motivates the application of other kind of techniques.

Order-preserving and locality-preserving hash functions, as well as SFCs, deal with this locality problem in some degree and are the most deployed in current solutions. By its utilization, indexed data by nodes retain *a certain data locality*, so that the system efficiency notoriously improves. Therefore, variating either mapping technique or application data domain, the performance of the system will differ from each other solution. This motivates the inclusion into the evaluation framework of both dimensionality and mapping approaches. By dimensionality we refer to the ability of the given solution to deal with either multi- or one-dimensional application data domains, and by mapping approach the way that the dimensionality reduction is performed. We do not consider the details of the mapping techniques regarding to type of information and domain because they are application-dependant and fall out of our study.

2.4.2 Quality of Service Criteria

Quality of service issues are differentiated into two sets. The first one considers evaluation parameters for the insertion and search operations. The last one considers the quality of the result set of the query resolution.

2.4.2.1 Insertion and Search Evaluation Criteria

In order to unify the terminology along the chapter, we do not make difference between data item or index, and so they are referred to as *data object* or *object*. Let us suppose now that a user is saving the information of its business and location, namely *data object*, into the distributed geographical service location application. The corresponding node inserts the data object into the system and, this way, it becomes available to all nodes. By doing this, we now consider three properties related to the processes from data insertions to query resolution. Firstly, should we consider the amount of storage used for each data object insertion? The amount of data object copies is considered because it measures the system's stress for data storage. Intuitively, storing a single copy of a data object would be sufficient to find it in future searches, and actually this is the best case. For instance, traditionally DHTs introduce data objects

only once. But either the specific application data domain or the SPN design can pose an obstacle, requiring to adopt a more complex approach. We summarize this into the **storage efficiency** property. Note that we do no consider replication and caching schemes in this study. Actually, they are applied to a wide variety of distributed and parallel systems, what includes the systems analysed in this chapter.

Another property included in the evaluation criteria is the **time efficiency**. The response time for data objects insertions and search resolutions becomes the most visible and detectable to the end user. Long-lasting operations cause desperation to human users and must be avoided. In particular, because we are evaluating similarity queries, we only account the time required for query resolution under time efficiency. Even when a solution reaches an adequate degree of storage and time efficiency, the solution can be unfair among nodes. For instance, some nodes could receive a greater amount of data objects than the rest, and some nodes could also be used for routing purposes more frequently than others. A solution will be more scalable when it provides both *data* and *routing* **load balancing**.

2.4.2.2 Evaluation Criteria of the Result Set

Now, let us suppose that a user has performed a range query q in our distributed geographical service location application example from a certain node, and this querying node is already in possession of the result set $\mathcal{O}'$. Let us suppose that we also know the exact set of data objects $\mathcal{A}$ expected to recover from q. How can we measure the quality of $\mathcal{O}'$? To this end we introduce two other properties: query **correctness** and **completeness**. Broadly speaking, a query q is resolved with correctness, when the result $\mathcal{O}'$ only contains data objects such that they were expected (i.e., all data objects from $\mathcal{O}'$ also appear in $\mathcal{A}$). Note that a correct $\mathcal{O}'$ may not contain *all* the expected data objects (formally $\mathcal{O}' \subseteq \mathcal{A}$).

As long as a query resolution provides the results with completeness, this ensures that $\mathcal{O}'$ contains at least all elements of $\mathcal{A}$. This allows however that some useless data object could be retrieved during the query resolution (formally $\mathcal{O}' \supseteq \mathcal{A}$). In such cases, querying node has to prune the results to leave only useful elements before passing them to the user. This is what we call a *user-side* prune. Otherwise, a *system-side* prune has detected all useless data objects and removed from result set (if any). We consider the way that information is pruned into the parameter **intersection**.

Moreover, it is easy to demonstrate that providing both *correctness* and *completeness* is a sufficient and necessary condition for a query to provide all possible results. That is, from *correctness* we have $\mathcal{O}' \subseteq \mathcal{A}$, and from *completeness* $\mathcal{O}' \supseteq \mathcal{A}$ (see Table 2.1).

Therefore, there is only a possibility that makes both conditions come true: $\mathcal{O}' = \mathcal{A}$. Nevertheless, when a system resolves queries providing both *correctness* and *completeness*, the operations could become too expensive either on time or storage.

2.4.3 Parallel Computing Evaluation

Broadly speaking, the consequence of query parallelization in a distributed setting is twofold: the query is resolved in a shorter response time, and the usage of system resources increases. In particular, given that SPNs perform node inter-communication by message passing, the bandwidth usage increases between all participanting nodes, as well as the usage of nodes' resources (e.g., computing cycles and main memory). Given that SPNs provide an inherent way of parallelizing tasks, we provide in this section an analysis on the parameters from the evaluation framework, which determine the feasibility and efficiency of the parallel computing. Parameters appear detailed in order of significance.

The most important factor is the system's **topology**. The way that nodes are organized dictates if operations can be performed in parallel and how. For instance, we have depicted two algorithms for range queries, depending on whether the topology provides path convergence or not (see Algorithms 2.1, 2.2). Remember that topologies with path converge provides no utility on query routing parallelization, since all sub-queries will travel along the same path.

Given a certain topology, systems can implement several **algorithms** to resolve a kind of query. That is, different search algorithms provide a solution to region- or sphere-based range queries, as well as to k-NN queries and spatial queries. It is easy to see that algorithm's efficiency and adequacy to the system greatly decide the performance of the query parallelization.

Some properties of the system limit also the performance on the query resolution. As it is stated before, **path convergence** prevents parallelization because it wastes useful resources. The amount of **data storage**, like the number of object copies, can provide several ways of accessing to the data, for instance. This facilitates that queries are parallelized. In addition, **time delay** in node inter-communication is not negligible. Implementing a parallel query resolution reduces the response time and increments the system's performance.

The last property we consider is the system's **load balancing**. The reader could see this property as uninteresting for the object of analysis. Actually, **data** and **routing load balancing** are very influencing. Whether data is uniformly stored into nodes or its distribution is skewed among nodes determines the query's response time, making

it stable or highly variable, respectively. Alternatively, routing load balancing will provide less overhead on query routing and, thus, less overhead on query processing to participant nodes.

Summing up, the reader should expect to see systems that greatly differ from each other. Not only the topology, but also a long list of system parameters influence the corresponding costs as well as the performance of the system when nodes cooperate.

2.4.4 Evaluation Criteria: Tuning and Terminology

All parameters introduced before are considered within the evaluation framework for all systems. Actually, systems considered into this chapter were designed so that, given a kind of query and an application context, they provide a trade-off between all these parameters. To put an example, a system that supports k-NN queries, could incur in more storage usage in only some peers (load unbalancing), in order to provide low response time (time efficiency) in system's operations, while the result set turns both correct and complete.

Table 2.2 shows the common measurements used along the analysis of systems providing similarity queries, in order to value the considered systems. Note that in both *storage* and *time efficiency* measurements, 1 is the best case. In other words, the best case for data storage is when only a data copy is introduced into the system (no replication or caching techniques are considered), and for time efficiency, when the response is at only one overlay hop. For the sake of simplicity, the denominator is only shown on the comparison tables.

Table 2.2: Tuning of the evaluation criteria for systems providing similarity queries.

Evaluation criteria	**Tuning**[1]
Topology	Descriptive name
Dimensionality	1-D | M-D
Mapping approach	1:1 | N:1 | M:P | N/M
Storage efficiency	$\frac{1}{\text{Num. data copies}}$
Time efficiency	$\frac{1}{\text{Num. overlay hops}}$
Load balancing	Yes | No
Correctness	Yes | No
Completeness	Yes | No
Intersection	System side | User side

[1] The term **N/A** can appear elsewhere when the tuning is not applicable for a given system.

In addition, before starting with the system evaluations, let us detail the common terminology we will use along this chapter. Note that every work uses its own nomenclature, but here for clarity to the reader, we unify them as much as possible to a common and simpler naming. The reader can find it in Table 2.3. Note that *selectivity ratio* expression refers to the ratio of the data domain that the query is selecting from. For instance, let us suppose an arbitrary range query with a selectivity ratio of 50% on the application data domain or on the i-th attribute. This means that the query is demanding for objects living on a half of the application data domain or the attribute domain, respectively. In the following sections, we delve into the comparison of different distributed systems.

2.5 Supporting Range Queries

After the first SPN appeared on 2001, researchers studied ways of enhancing SPNs by supporting high-level queries. The first kind of search abstraction we study is the range query. We present in Table 2.4 the measurements from different remarkable systems, using the notations from Table 2.3. Systems appear in ascending order from the publication year. This provides an outlook on the system design evolution. Hereafter we proceed with the study by classifying their topology into two big sets, namely **flat** and **hierarchical** SPNs. Differentiating systems from their topology will help to a better understanding of their lookup algorithms and performances.

Table 2.3: Common terminology along the evaluations of systems providing similarity queries.

Term	Description
N	Number of nodes within the network
M	Number of attributes of the application data domain
M_D	Number of attributes included in the data object
M_Q	Number of attributes included in the query
S	Total query's selectivity ratio
S_i	Selectivity ratio of the i-th attribute
K	Number of nodes to visit to complete the query
K_i	Number of nodes to visit for the i-th attribute during the query resolution
k	Number of data objects to retrieve in a k-NN query
fo	Stands for *fan-out* (i.e., number of outgoing links in the routing table)

As the counterpart, it is well-known from software engineering field that an algorithm complexity is tightly influenced by the data structure supporting it. SPNs are not an exception. A SPN can be seen as a distributed data structure, so that every node is responsible of only a part of it. Thus, the way nodes are distributed within the network and the connectivity between them will determine the cost of the query resolution.

2.5.1 Flat Systems

The works considered here employ either a **ring-based** or a **grid-based** topology. Thus, algorithms cannot be applied exactly to each other. This motivates their further classification into this study.

2.5.1.1 Ring-based Topology Systems

The following systems, MAAN [30], SkipNet [26], Mercury [29] and M-Chord [49], distribute nodes in a ring topology. Systems aim at providing a time- and resource-efficient query resolution. This motivates that data objects should be strategically placed within the network. Ring-based topologies (see Fig. 2.6a for an example) provide an easy way of iterating among successors and/or predecessors. This becomes a powerful and efficient mechanism of communication whenever the system stores data objects contiguously within the SPN keyspace $\mathcal{I}$. Different *mapping approaches* $\mathcal{F}_{\mathcal{O}}$ appeared to this end. Any given mapping technique has the challenge of placing contiguous data objects in the data domain, contiguously within the ring. To do so, most of mapping functions are order-preserving hash functions, so that if $\mathcal{F}_{\mathcal{O}}$ is the function and x, y two data objects, whenever $x \leq y$, $\mathcal{F}_{\mathcal{O}}(x) \leq \mathcal{F}_{\mathcal{O}}(y)$ succeeds.

A sketch of the algorithm used by all these systems is detailed in Algorithm 2.2 and depicted in Fig. 2.6a. A querying node starts the search. Let us assume that the query $q = [lb, ub]$, $lb \leq ub$, is asking for all objects between the lower bound lb and the upper bound ub. During the first routing phase, the query is directed to the *corresponding node* that owns one of these bounds, namely $\mathcal{F}_{\mathcal{O}}(lb)$, and thereafter, the query is spread among all nodes responsible of the range $[\mathcal{F}_{\mathcal{O}}(lb), \mathcal{F}_{\mathcal{O}}(ub)]$. For the sake of clarity, the figure depicts a sequential order on the node visiting process, but more complex algorithms (e.g., [50]) can be applied to achieve that in a more time-efficient way. See Section 2.5.3 for a discussion on how it can be implemented. Now, let us see what systems do.

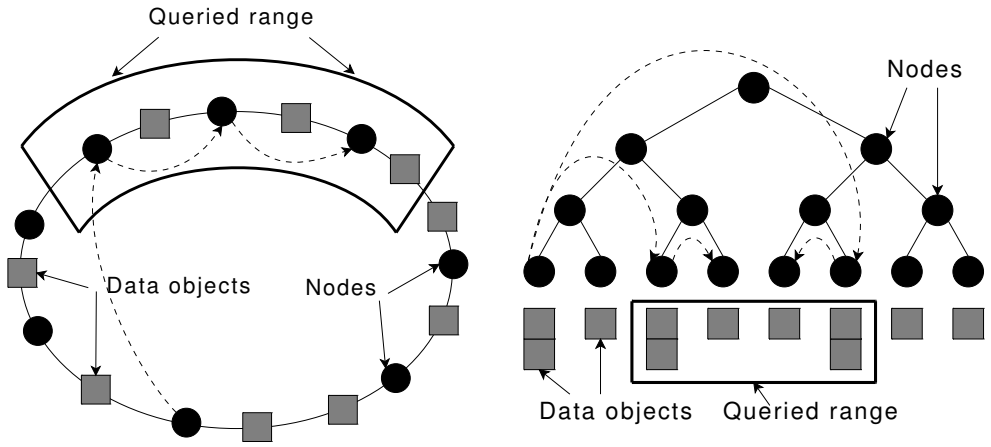
The time efficiency column in Table 2.4 depicts clearly the main two phases on the range query algorithms. SPNs studied here locate a node in a logarithmic number of

hops with regards to the number of nodes in the worst case and, so, the cost formulation includes a first operand $(\log N)$. The second operand corresponds to the cost of spreading the query within the queried range. Generically, this part is considered to be the number of nodes (and so single hops) that are necessary to visit to answer the range query. To put an example, SkipNet has a response time of $O(\log N + S \times N)$. The logarithmic cost is for the first part and $S \times N$ corresponds to the time needed to visit the set of nodes responsible of the queried range. S means the *selectivity ratio* of the range query, and we assume that all nodes $\mathcal{P}$ are uniformly distributed at random.

There are two particular systems with a ring topology, MAAN and Mercury. They show a more complex formulation of the time efficiency. This is because both of them map every single dimension from a M-dimensional data domain into the 1-dimensional $\mathbb{I}$. This motivates the increase on the data storage requirements. Above all, they are in essence the same kind of approach. The difference lies in which MAAN consists of one ring, responsible of all dimensions of the data domain. Instead, Mercury employs as many rings as dimensions is considering the application data domain. Rings in Mercury become clusters of nodes sharing information of the same dimension.

These systems employ an order-preserving hash function to determine the responsible node for a given data object. Instead, M-Chord uses iDistance [48] and this mapping approach behaves in a different way. Adopting iDistance requires of few steps. 1) Data objects are sampled before the network creation in order to find out n data clusters. And 2), for every non-overlapping data cluster, a centroid data object and the radius to the farthest object from the same cluster are set up, and become *global knowledge* for all nodes. Therefore, using iDistance is inappropriate in applications where there is a little or no knowledge of the dataset, and because the system needs of global knowledge, that should be avoided in large scale distributed systems.

In addition, by using the iDistance mapping, range queries are proceeded as in the sphere-based approach. Given that data objects are clustered, a given range query may overlap partially or totally areas from different data clusters. iDistance transforms any overlapping area into a contiguous segment on $\mathbb{I}$, but all partial segments may not be contiguous among them. This motivates that the range query resolution is parallelized in systems based on iDistance mapping, and in particular in M-Chord. The querying node, then, identifies the queried keyspace segments and sends as many messages as number of segments. The two phases approach still remains here, but for any individual segment. Although M-Chord perform a parallelized range query, the time efficiency remains similar to the rest ones (surely with a smaller hidden constant),



(a) Ring-based range query algorithm (adapted from [29]).

(b) Tree-based range query algorithm (adapted from *shower* algorithm from [31]).

Figure 2.6: Approaches for range query algorithms according to the topology.

and in addition incurs in a greater network overhead. For further details on iDistance, the reader is kindly forwarded to [48].

2.5.1.2 Grid-based Topology Systems

SWAM-V [27] and HyVonNe [51] are based on a grid topology. In particular, they place nodes into a Voronoi diagram, and then the dual Delaunay triangulation organizes the neighbors of nodes. HyVonNe selects a *leader* for every grid's region in order to make it more flexible to node movements and topology changes. Alternatively, SWAM-V adds some other links to far away nodes to reduce the time latency down to a logarithmic cost. Given that this routing cost is a lower bound for both systems, we only include the full study of SWAM-V.

The key difference between SWAM-V and the other flat ones is the following: nodes in SWAM-V share the information they have, instead of inserting it into other nodes. This fact motivates the organization of nodes according to the content similarity among nodes, and a particular design of the lookup algorithms. The two phases lookup approach remains still valid. The difference resides in that (hyper)sphere-based range queries are considered, instead of (hyper)rectangle or hypercube. This way, the first step locates a node responsible for a central point p; in the second step the query is spread to the nearest nodes (delimited by a time-to-live), and this determines the queried *radius*. Reader may have observed that this approach does not provide *completeness* to the result, given that some farther data objects can exist re-

lated to the range query. And actually this is what happens. But, by definition of the sphere-based range query (refer to Definition 2.4), the result set is complete whenever the whole given radius is visited.

2.5.2 Hierarchical Systems

A hierarchical system organizes nodes in the shape of a pyramid, with each row or level of nodes linked to other nodes beneath them. For the purpose of our analysis, we relax its definition by allowing nodes appear either at any level on the hierarchy or only in its leaves. In fact, the hierarchy still remains even though only in a logical level.

Compared to flat topologies, most of hierarchical ones provide a natural way of performing lookup operations in a parallel way, instead of a sequential resolution. In other words, the hierarchical structure provides different paths where to spread the search operation from the very beginning, namely from the querying node (see Algorithm 2.1 and Fig. 2.6b for an example). As it is stated in Section 2.4.3, there exists a trade-off between network overhead and time efficiency: This approach incurs in more overhead, for the greater bandwidth usage to say the least, so that the whole result set is retrieved from the system within a shorter response time. Unless noted, all shown time measurements for hierarchical systems consider a parallel lookup resolution.

The included hierarchical SPNs in this part of the study are broadly classified into two topological groups: **tree-based** and **super-peer-based** approaches.

2.5.2.1 Tree-based Topology Systems

The tree-based topologies are the following ones: Trie-based [31], DST [52], Skip Tree Graphs [53] and SDI [54]. These systems (except DST) place nodes into the leaves of the tree. Then, the logical built tree specifies the interconnection between nodes and, thus, determines the time delay for lookup operations. See Fig. 2.6b for a generic example.

Trie-based constructs a load-balanced trie, where all nodes are responsible for the same amount of information. The drawback here is that nodes require from prior knowledge of the dataset distribution to correctly built the trie. One of the strengths is that Trie-based is designed from the very beginning for a real environment, so nodes are placed strategically within the trie to provide structurally data replication. Trie-based provides two algorithms for range queries. The time efficiency shown is for

the *min-max traversal* algorithm, that consists exactly in the sequential, two phases approach explained earlier. An improved approach, called *shower* algorithm, is also presented and consists in a parallel query resolution, very similar to that shown in Fig. 2.6b. This case, the resolution time decreases down to a logarithmic cost, at the cost of a greater network overhead.

Skip Tree Graphs become an abstraction for Skip Graphs/Nets, that provide a tree-like navigation of the distributed data structure. Because Skip Tree Graphs are isomorphic to Skip Graphs/Nets, Skip Tree Graphs are considered to provide the same costs like in SkipNet. Though the hidden constant in the time efficiency is smaller than in SkipNet, Skip Tree Graphs suffers from greater delays at insertion times.

SDI is a particular tree-approach where the navigation, differently from the systems before, is performed in a bottom-up approach. All lookup operations in SDI are inherently parallel, so that the time delay is reduced to a logarithmic cost. SDI, unlike Trie-based, Skip Tree Graphs and DST, does not utilize any mapping function to data objects, but uses the application data domain to organize information into nodes.

There exist a family of systems based on data-driven tree abstraction built upon DHTs, where PHT [55] (Prefix Hash Tree) embodies this family. PHT builds a binary tree link structure among nodes, so that nodes storing contiguous data appear connected. Moreover, this structure is said data-driven because data insertions determine the tree organization. PHT provides data load balancing because it is based on the default DHT functionality ($put(key, value)/value \leftarrow get(key)$). But PHT's penalty is the great overhead in communication cost. PHT resolves range queries iterating over a logical contiguous data domain, but actually along far away nodes from each other. This augments the total overhead on the routing cost compared to other systems with data locality, providing a poor solution in terms of routing efficiency.

Actually, we include the analysis of DST (Distributed Segment Trees) as a lower bound of PHT and PHT-based systems. DST provides lower time delays than PHT and this is because of the structural data replication performed in DST. Every object insertion is performed in average into $O(\log N)$ nodes. This fact, in addition to the inherent parallel lookup resolution, provides a logarithmic resolution response time in terms of hops. The drawback is that this design clearly does not provide load balancing. Firstly, because higher nodes in the tree receive more queries because of the top-down tree routing. In addition, DST insertion algorithms tries to balance the data load among nodes by defining a maximum storage threshold. Unfortunately, it is easy to see that higher nodes in the tree will reach sooner the threshold than leave nodes, producing a data load unbalancing among nodes.

2.5.2.2 Super-peer-based Topology Systems

Three systems based on a super-peer topology [56] are considered: the work of Liu *et al.* [57] (namely LET), EZSearch [58] and JXTA Search [59]. The main key on the design of a super-peer topology is that there are two types of nodes: normal peers and super-peers. Broadly speaking, normal peers store information and are connected to a super-peer. The role of super-peers is indexing information of connected normal peers, in order to speed up lookup resolutions. Super-peers are inter-connected in a way so that super-peers can reach each other at low delay. Evaluating the fairness of the system, it is easy to see that super-peers receive more communication than normal peers and stored information. Thus, super-peer-based designs does not present good qualities in terms of load balancing (as it can be seen in Table 2.4).

These systems face the problem from two different viewpoints: whilst in EZSearch data is inserted into the system, super-peers on LET and JXTA Search (based on JXTA [60]) build the indexes of the information that their normal peers possess. In addition, while LET and JXTA Search present a two-tier hierarchy (normal peers below and super-peers above), EZSearch (based on ZigZag [46]) builds a balanced tree of variable height, so that each leave cluster has approximately the same amount of peers. Super-peers in EZSearch (called *head* and *associated-head*), can appear in more than one level in the hierarchy. This topological differences clearly determines the time efficiency on range query resolutions.

In these systems, any node can start the range query, and it is directed to a *corresponding super-peer*. In LET and JXTA Search, the node's super-peer is the *corresponding super-peer* and it is responsible of checking into its indexes whether the query can be resolved locally in the cluster, or the super-peer needs to contact to other super-peers. This *global* knowledge is updated between super-peers by exchanging periodically summaries of their indexes. This way, the query forwarding is more efficient and unnecessary super-peers are not visited. The operation concludes when the asked super-peers send to the *corresponding super-peer* their results and then, all them are forwarded to the querying peer. Note that in JXTA Search nomenclature, super-peer nodes are called *hubs*, normal peers are differentiated between *information providers* and *consumers*, and clusters are called *groups*.

In EZSearch, range queries are directed to the *corresponding super-peer* of the corresponding cluster. Unlike LET and JXTA Search, EZSearch makes the assumption that there is a necessary population of nodes in leave clusters, so that all data objects for the range query will live there. This assumption is guaranteed by correctly setting up the z-factor. This factor determines the number of nodes within clusters and, thus, the

height of the tree. The goal is clear: to avoid visiting other close clusters, that would introduce longer response times. Note that the time efficiency formulation also accounts the z-factor, and that cost can be interpreted as the number of hops required to contact the *corresponding super-peer*.

2.5.3 Range Query Evaluation: Conclusions

Most of systems interpret range queries as a *region-based* search instead of the *sphere-based* approach. This is because with the first one, systems can guarantee that whenever there are some data objects lying within the queried range, they will be retrieved. But, actually, these two interpretations of range queries address different problems. For the sake of clarity, while a region-based approach is necessary for geographical location service, in order to provide all elements from the given region, the same interpretation cannot be used in the case of Information Retrieval (IR) applications, or in scenarios where nodes contribute with their own content without inserting it into the system.

Actually, the sphere-based range query can be seen as a k-NN query with a fixed system-wide radius. This is very useful in certain application domains where data object indexation is not so strict, but flexible. For instance, documents in IR applications are indexed according to document' meta-data, and nodes supporting their own content can provide a variable number of any kind of data objects. For these scenarios, a region-based approach is not practical in terms of number of peers to be visited and, thus, response time: Almost all nodes would be candidate for visiting along the query resolution. This means that every query should be broadcasted to all nodes, which is not suitable in the large scale. Thus, a sphere-based range query deals with the uncertainty problem and provides a feasible solution.

The quality of sphere-based approach depends on two factors: 1) the quality of information and/or node clustering achieved by the system, and 2) the query radius that determines the area to search into. Both of them depend on the specific system design (see SWAM-V [27] or the work of Liu *et al.* [57] for some examples). Unfortunately, there is a factor that systems cannot govern: the dataset distribution. This factor informs about the data clustering degree. For this kind of systems, highly clustered data objects will provide inherently better results than a dataset that is uniformly distributed within the data domain.

Most of systems, instead, insert data objects into the system and they are stored by the responsible nodes, selected deterministically. This way, an exact match is resolved in the same way than in the insertion case. For these systems the dataset distribution

becomes also a key issue. A uniformly distributed dataset will provide inherently good data load balancing among nodes. Otherwise, in most of the SPNs, the system will suffer from data load unbalancing, given that some nodes will support more data objects than others. Actually, this problem is an open issue for most of the SPNs.

We have also explained that hierarchical systems provide in most of them a natural, inherent way of parallel lookups. What about flat topologies? There exists an important reference in this field. The work of S. El-Ansary *et al.* [50] introduces a broadcast algorithm for ring-based topologies with a total time delay of $O(\log N)$. This algorithm takes advantage from the nodes' knowledge. Every node in a ring topology knows the fragment of the keyspace $\mathbb{J}$ it is responsible for, as well as the keyspace fragments of its neighbors. This way, all nodes are contacted by the broadcast message in the worst case with a time delay of $O(\log N)$. It is easy to see that a variation of this algorithm could also be applied for the case of range queries. During the second phase of the range query, a set of contiguous nodes are responsible of the queried range and must be visited. Thus, the first visited node could start a range-restricted broadcast, or multicast, based on the same technique. Actually, we will consider this design in our proposed range query services under Chapter 4. For instance, SkipNet has a time delay of $O(\log N + S \times N)$. The SkipNet improvement would be from $S \times N$ to $\log(S \times N)$, reducing the response time down to $O(\log N + \log(S \times N))$. However, the trade-off still exists here. Less time delay means greater network overhead.

To conclude this section, it is worth noting that in last years more and more systems aim at providing enhanced lookup services by means of hierarchical SPNs, as it can be seen from last rows in Table 2.4. This is motivated from the fact that most of the hierarchical designs provide inherently parallel query resolution. As we have discussed early in Section 2.5.2, a parallelized algorithm is not for free, but has the cost of more bandwidth usage and busy nodes per unit time. Nevertheless, given that nowadays Internet connection bandwidth and computer resources are growing rapidly, bandwidth and computation power are considered as not so restrictive as some years ago. As it is seen from our analysis, this opens a door to systems that are not so efficient in resource consumption but effective in reducing response time.

Table 2.4: Evaluation of systems on range queries.

System	Topology	Dimensionality	Mapping approach	Storage Efficiency	Time Efficiency	Routing Load Bal.	Data Load Balancing	Correctness	Completeness	Intersection (side)
JXTA Search [59]	Tree (Super-peers [56])	1-D	N/M	1	$\Psi(N, \|C\|, S)^1$	No	N/A	Yes	Yes	System
MAAN [30]	Ring (Chord [7])	1-D	1:1	$O(M)$	$O(\log N + K)$	No	No	No	No	User
		M-D	1:1	$O(M)$	$O(\sum_{i=1}^{M} (\log N + S_i \times N))$	No	No	No	No	System
SkipNet [26]	Ring (Skip Graph [61])	1-D	1:1	1	$O(\log N + S \times N)^2$	Yes	No	Yes	Yes	System
Mercury [29]	Multi-Ring (Symphony [13])	M-D	1:1	$O(M_D)$	$O(\sum_{i=1}^{M_Q} (\frac{1}{f_0} \log^2 N + S_i \times N))$	Yes	Yes	Yes	Yes	System
SWAM-V [27]	Small-World (Voronoi diagram)	M-D	N/M	1	$O(\log N + S \times N)$	No	N/A	Yes	Yes	System
Trie-based [31]	Binary Trie P-Grid [14]	1-D	1:1	1	$O(\log N + S \times N)^3$	Yes	Yes	Yes	Yes	System
Liu *et al.* [57]	Tree (Super-peers [56])	M-D	N/M	1	$\Psi(N, \|C\|, S)^1$	No	N/A	Yes	Yes	System
M-Chord [49]	Ring (Chord [7])	M-D	M:1	1	$O(\log N + S \times N)^2$	No	Yes	Yes	Yes	System
DST [52]	Binary Tree	1-D	1:1	$O(\log N)$	$O(\log N)^4$	No	No	Yes	Yes	System
Skip Tree Graphs [53]	Tree + Skip Graph [61]	M-D	M:1	1	$O(\log N + S \times N)$	Yes	No	Yes	Yes	System
EZSearch [58]	Tree (ZigZag [46])	M-D	N/M	1	$O(\log_z N + C_N)^5$	No	No	Yes	Yes	System

Continued on next page

Table 2.4 – *Continued*

System	Topology	Dimen–sionality	Mapping approach	Storage Efficiency	Time Efficiency	Routing Load Bal.	Data Load Balancing	Correctness	Completeness	Intersection (side)
SDI [54]	Tree	M-D	N/M	1	$O(\log N)^4$	Yes	No	Yes	Yes	System

[1] The cost depends on the number of nodes into the system N, the number of clusters $|C|$ and the query selectivity ratio S.
[2] Costs inferred from the search algorithm/system description.
[3] Sequential, two phases resolution cost. The cost of the corresponding parallel algorithm is $O(\log N)$.
[4] The presented costs are from the parallel resolution algorithms.
[5] EZSearch makes the assumption that a query is resolved by visiting a single cluster. Here, C_N refers the number of nodes into that cluster.

2.6 Supporting k-NN Queries

k-NN queries provide up to k data objects that are the most alike to a given object. The reader can find its meaning at Definition 2.6 and in Fig. 2.4b an example. Besides, the basic algorithm is detailed in Algorithm 2.3. A querying node performs the k-NN query by specifying a pair $q = \{p, r\}$ and k. p is a point or data object and r defines an initial radius, where k sets the maximum number of objects to collect.

It is easy to see that sphere-based range queries and k-NN queries are very similar. In fact, the difference lies with the value set to the radius r. While in sphere-based range queries it takes a system-wide, fixed value, the value of k is readjusted in k-NN queries (lines 8,17 of the Algorithm 2.3). In consequence, any solution that adopts a sphere-based range query approach, can also resolve k-NN queries easily after minimal modifications to the range query algorithm. In particular, most of the systems included in this section appeared also in the range query evaluation section. Some other systems are appended.

It is worth noting that most of time efficiency measurements of Table 2.5 have the shape $O(\log N + K)$. The first part $O(\log N)$ considers the time to reach the *corresponding node*. From Table 2.3, K means the number of nodes required to be visited to complete the query, but where its value is hard to quantify formally. Thus, K express a supplement in the response time on the query resolution. Some factors influence K's exact value: (i) the system design, (ii) the number k of elements to retrieve, and (iii) the specific situation of the network while the query is performed. The last factor includes either the node organization, as well as the amount of data stored in nodes. It is easy to see that while less data objects are stored in nodes, more nodes will likely be visited. Similarly, when the value of k increases, the probability to visit a higher number of nodes also augments.

Hereafter we proceed with the study of a remarkable set of systems, differentiating between **flat** and **hierarchical** SPNs. The topology greatly influences into the algorithm design and its performance.

2.6.1 Flat Systems

We broadly differentiate the systems between **ring-based** and **grid-based** topology systems. Algorithms and techniques are different from each other.

2.6.1.1 Ring-based Topology Systems

We consider in this section the following systems: M-Chord [49] and AON [62]. M-Chord has been discussed in the section before. AON (that stands from Attributed-based Overlay Networks) supports both range and k-NN queries. The motivation to include it into this study is for the given similarity to Mercury. Both systems have essentially the same network structure: one ring for every dimension. The difference lies with the data management. Data objects in AON are neither inserted nor mapped, while nodes in Mercury insert data objects into the network after their mapping. This provides AON for a data-driven ring construction. Nodes in AON are organized into attribute-based rings. The data objects that nodes store locally, specify the node's place into the ring.

For fast navigation through the ring, AON nodes have a routing table per ring that resembles a Chord finger table. Even though the network structure seems to provide a good infrastructure for efficient query resolution, it poses some inconvenients. Whenever the local information of nodes become dynamic and highly variable, the placement of nodes into the different rings they are participating in will vary accordingly. To do so, nodes employ a leave-join mechanism. In consequence, not only the network can suffer from churn (i.e., nodes leaving and joining the network), but also along the time a node is present into the system, since node's content will certainly vary. Another inconvenient is the amount of node state AON and Mercury require for an efficient routing into all rings that nodes are participating in. AON only presents an evaluation for 1-dimensional scenario, but it is said that multi-dimensional data domains can be also supported.

2.6.1.2 Grid-based Topology Systems

This part of the k-NN study considers SWAM-V [27], a Voronoi-based node organization. The algorithm applied here is the same like in the case of range queries, but slightly modified to readjust the radius r in order to complete the query with k data objects.

Besides, we study two systems based on CAN [11]. They are pSearch [19] and M-CAN [63]. Both systems have a common design decision: reducing the data domain dimensionality (M) to that of the underlying CAN (P). Moreover, they avoid the application of some dimensionality reduction function $\mathcal{F}_{\mathcal{O}}$. This makes easier the use of CAN and accelerates all operations by preventing the use of mapping functions.

pSearch is based on eCAN [23], an improved version of CAN that achieves a worst case logarithmic communication cost between any pair of nodes. pSearch is intended for Information Retrieval applications, and introduces two algorithms based on eCAN and Information Retrieval techniques. One is called *pVSM*, for the case of Vector Space Model, where the m most descriptive terms from documents are taken. The other one is called *pLSI*, for the case of Latent Semantic Indexing, which reduces the dimensionality of VSM technique to l, $m > l$, by applying semantic reduction techniques. The reader can observe in the time efficiency formulation a variable cost $\zeta(k, m)$. This cost summarizes the number of nodes needed to be visited by an arbitrary k-NN query, that highly depends on the number of elements to retrieve k and the size m of the vector space model. When Latent Semantic Indexing is used, the delay becomes $O(\log N + \zeta(k, l))$.

M-CAN employs the iDistance [48] mapping technique to index all data objects into CAN. M-CAN is then analogous to M-Chord, and the whole M-Chord discussion can be applied to M-CAN. Note that the time efficiency accounts the communication cost based on CAN. It would be easy to replace CAN by eCAN, so that the time efficiency would be reduced to a logarithmic cost $O(\log N + K)$.

2.6.2 Hierarchical Systems

Under the umbrella of hierarchical systems we look at two tree-based topology systems, PIRD [64] and SDI [54], as well as two super-peer based topology systems, namely Liu *et al.* [57] and EZSearch [58]. All systems except PIRD have been discussed in the range query section. Given that these systems employ a sphere-based range query approach, they modify the query algorithm to successively adjust the radius r in order to support k-NN queries. Let us put the focus on PIRD.

PIRD is based on Cycloid [65] and structurally provides geographical k-NN queries, where the result set contains indexes to data objects that are located in nodes close to the querying node. This way, the querying peer will accelerate the last peer-to-peer transmission. To do so, Cycloid introduces a two-layer indexation. The first layer organizes nodes and indexes into different clusters, and the second layer (i.e., at any of the clusters) organizes information according to the geographical proximity. The proximity is presented as a distance vector obtained from a sampling to a set of landmarks. This way, the distance evaluation is performed by comparing those vectors. Nodes also use these vectors to locally cluster the indexes according to the distance.

The query algorithm takes advantage of this indexing to retrieve the k closest data objects to the query, and also by retrieving them from the geographically nearest nodes to the querying node.

Nevertheless, the indexes are inserted in a system-wide number of times (L), incrementing the data storage on nodes. To do so, PIRD builds L different keys in a way that similar data objects will be assigned to similar keys with high probability. This can be seen also as to having similar data objects indexed in the same node. Queries are spread to L different keys, and the querying node has the last responsibility to return a valid result set of k data objects from those L partial sets received from queried nodes. Lastly, the time efficiency shown for PIRD is the time for a single lookup operation in Cycloid, because the k-NN query is parallelized.

2.6.3 k-NN Query Evaluation: Conclusions

It is easy to see a common denominator within the study of systems supporting k-NN queries. Because of the nature of k-NN query algorithms, systems that also supports range query in a sphere-based approach prevail over the others. Actually, this section does not include any system that supported range queries in a region-based approach. This is because both approaches are quite distant from each other, and pose different challenges during the system design.

Another common characteristic among systems is the two phases approach. Remember that in the first phase the *corresponding node* is reached, while in the second one close nodes are accessed, in order to build a result set. This resolution mechanism is dictated by the definition of the k-NN query. There are only few exceptions to this though. One example is the innovative design of PIRD [64] that, by means of several keys per data object, the system achieves a per-node index clustering, so that the query resolution is performed locally at several *corresponding nodes*.

Another family of systems that perform only data indexation are *data networks*. Nodes in data networks hold their own information and only indexes are built throughout the system. The way the indexation is realized will decide whether the two-phase approach remains still valid, or ad-hoc algorithms will have to face the problem. An example within the family of data networks is the work of Liu *et al.* [57]. Unless in hierarchical systems supporting range queries, there is not much flexibility for systems to totally parallelize k-NN queries. That is, with a perfect knowledge of nodes' content, querying node would be able to direct the query to the corresponding nodes. However, this requires of global knowledge, what is not feasible on the large scale. A slight longer response time is the low price users have to pay for.

Table 2.5: Evaluation of systems on k-NN queries.

System	Topology	Dimensionality	Mapping approach	Storage Efficiency	Time Efficiency	Routing Load Bal.	Data Load Balancing	Correctness	Completeness	Intersection (side)
pSearch [19]	Hypercube (eCAN [23])	M-D	M:P	1	$O(\log N + \zeta(k,m))$[1]	Yes	Yes	No	Yes	System
SWAM-V [27]	Small-World (Voronoi diagram)	M-D	N/M	1	$O(\log N + K)$	No	N/A	Yes	Yes	System
Liu *et al.* [57]	Tree (Super-peers [56])	M-D	N/M	1	$\Psi(N, \|C\|, k)$[2]	No	N/A	Yes	Yes	System
M-Chord [49]	Ring (Chord [7])	M-D	M:1	1	$O(\log N + K)$	No	Yes	Yes	Yes	System
M-CAN [63]	Hypercube (CAN [11])	M-D	M:P	1	$O(P\sqrt[P]{N} + K)$	Yes	Yes	Yes	Yes	System
AON [62]	Ring	1-D	N/M	1	$O(\log N + K)$	No	N/A	Yes	Yes	System
PIRD [64]	Hierarchical (Cycloid [65])	M-D	M-P	$O(L)$	$O(d)$[3]	No	No	No	Yes	User
EZSearch [58]	Tree (ZigZag [46])	M-D	N/M	1	$O(\log_z N + C_N)$	No	No	Yes	Yes	System
SDI [54]	Tree	M-D	N/M	1	$O(\log N)$[4]	Yes	No	Yes	Yes	System

[1] The final delay is related to $\zeta(k,m)$, where k is the k-NN factor, and m is the number of terms in the Vector Space Model.

[2] The cost depends on the number of nodes into the system N, the number of clusters $|C|$ and the number of objects to retrieve k.

[3] Queries in PIRD are parallelized, where $O(d)$ is the cost of a single lookup.

[4] The presented costs are from the parallel resolution algorithms.

2.7 Supporting Spatial Queries

Spatial queries can be seen as a particular case of range queries, either in a sphere-based or region-based approach. Supporting spatial queries means that the systems must support operations in at least two dimensions, which specify the geographical location of objects, for instance by a pair {latitude, longitude}. In addition, these systems can support either point data objects or rectangular data objects. The former describes the data object's location by a single point ($\{x, y\}$), while rectangular ones are extended objects that occupies an area. Rectangular data objects are usually specified by a pair of points $\{x1, y1\}, \{x2, y2\}$ which describe the minimum bounding rectangle of the covered area. To give some examples, a restaurant's location is seen as a point data object, while the area where a superstore makes home deliveries can be described by a rectangular area.

When users perform spatial searches, the systems have to provide data objects that are located in or overlap the specified area of arbitrary size, while the search area is usually specified by the minimum bounding rectangle $\{x1, y1\}, \{x2, y2\}$. It is easy to see that the challenges of this kind of systems are particularly different from those ones supporting range queries, what has motivated their study separately.

This section goes on with the evaluation of significant systems supporting spatial queries, by differentiating between **flat** and **hierarchical** systems. All systems included here can be found in Table 2.6, listed in ascending order of the publication year.

2.7.1 Flat Systems

We consider two systems in this section, SDS [66] and Spatial Query [67]. SDS is a layered design, using Chord as a distributed infrastructure to support a quadtree. Given that the emphasis is put on the quadtree design, SDS is further detailed in Section 2.7.2.

Spatial Query is designed onto CAN [11]. CAN naturally supports multi-dimensional domains, so that little effort is required to use CAN for supporting spatial queries. Spatial Query supports both point and rectangular data objects. Thus, the focus is moved to guarantee data load balancing through nodes, what is addressed in two ways. A new node selects randomly to join into an area with high data density, and splits it into two sub-areas. Secondly, by limiting the number of times a region can be split. This way, nodes are always responsible of a minimum area. This approach has a double effect though. Firstly, nodes in Spatial Query are distributed more uniformly because of the threshold. But conversely, nodes responsible of areas very populated

that have reached the threshold, have no way of balancing the data load with other nodes.

2.7.2 Hierarchical Systems

Hierarchical systems are classified in this section into two topological types: tree-based and super-peer-based systems.

Tree-based Topology Systems

We analyse two tree-based topology systems, DHR-Tree [68] and SDS [66]. Both of them present a distributed design of already existing tree data structure.

DHR-Tree, that stands from Distributed Hilbert R-Tree, is based on P-Tree [69] and distributes nodes through a Hilbert R-Tree. Mainly, a Hilbert R-Tree uses the Hilbert [35] space filling curve to produce the so-called *Hilbert values*, and use them to index data objects into a B+-Tree. The idea behind this distributed design is to use the Hilbert values also as node identifiers, thus operating as data nodes in a B+-Tree, and to use the B+-Tree as nodes' routing table. The B+-Tree is populated with data that nodes locally host and the B+-Tree is partially built on every node. Thus, data updates may alter the minimum bounding rectangle a node is responsible for, and so forth on nodes at higher levels in the B+-Tree. The time efficiency in this scenario is tight to the order d of the R-Tree and to the search algorithm of two phases.

SDS (Spatial Data Service) does not build an SPN, but employs Chord as its underlying communication network. In addition, SDS builds multiple indexes for data objects. Thus, the focus is put on balancing the data load among nodes and structurally avoiding communication between neighboring nodes for updating minimum bounding boxes. To do so, nodes consider the whole data domain managed by a MX-CIF Quadtree [70]. The centroids of every region in the quadtree become, after applying a consistent hash function $\mathcal{F}_{\mathcal{O}}$ (like SHA-1), the key to route within Chord. This way, either rectangular data objects or spatial queries are processed locally as in the quadtree. The result of this process is the set of regions in the quadtree where the data object should be indexed or the search be performed, respectively. The next step is routing the corresponding messages to the keys produced by the centroids of the set of regions. Thereafter, nodes responsible of the keys will perform the corresponding operation.

SDS balance the data among nodes by applying two techniques. The first one is the use of a consistent hashing function $\mathcal{F}_{\mathcal{O}}$ to the centroids. The second one is by

ensuring that data objects are indexed into the quadtree only under a certain level (so-called f_{min}) of the tree. Thus, f_{min} dictates the trade-off between load balancing and indexing storage. When spatial queries are performed, the same data index can be retrieved several times from SDS because there is no coordination between nodes to collect a common result set. It is easy to see that this produces a non-negligible network overhead. SDS faces the problem of duplicate data transmissions by limiting at owner nodes that querying nodes could download only once each requested data object. Further requests are discarded. Even though the approach is nice, but quite naive, the time efficiency for spatial queries is reduced to a logarithmic cost. Nevertheless, the query is parallelized into several keys per query, suffering from more network overhead.

Super-peer-based Topology Systems

This section considers two systems: Liu *et al.* [57] and Globase.KOM [20]. Given that the work of Liu *et al.* has already discussed in prior sections, we center the study on Globase.KOM and the comparison between them. Both systems employ a similar model of super-peer network, where super-peers maintain an index of all the content that their normal peers hold. Globase.KOM defines a node identifier with three separated parts: the $\{lat, lng\}$ coordinates of the node's location, the area that the node is responsible for, and a random part to allow several peers in the same geographical area.

A key difference between the work of Liu *et al.* and Globase.KOM is the number of levels in the hierarchy. While Liu *et al.* define a 2-layer network structure, Globase.KOM is organized in several layers so as not to overload super-peers. When a super-peer sp_1 is overloaded, a new sub-cluster is built under sp_1 and a new super-peer sp_2 is elected. The effect is that a set of normal peers that were children of sp_1 are now children of sp_2, and sp_2 remains as a child of sp_1, thus balancing the load among super-peers. The network construction algorithm produces a network structure of super-peers and normal peers that resembles to that of an R-Tree. This motivates that the response time for query results is logarithmic in the number of clusters, plus the cost of asking to normal peers. Note that searches are performed in parallel, thus increasing the bandwidth usage.

2.7.3 Spatial Queries: Conclusions

Spatial queries pose additional challenges to systems, like supporting rectangular data objects, even though not all above seen solutions support them. Actually, systems are

tuned up to improve their performance in the given geographical application context.

Besides, there is an evolution on the system designs from those supporting range queries to these ones supporting spatial queries. Even though range queries and spatial queries have a lot in common, hierarchical systems are significantly predominant in the study of spatial queries while the same is not true for the study of range queries. This is motivated because a hierarchical approach suits better than a flat one for supporting spatial queries.

Another important issue addressed by Globase.KOM is that data objects and nodes are indexed/placed in the geographical area they belong to. This is a very interesting approach that provides at the same time data locality and consistency to the system against system failures. This is a property that is also considered in our geographical information service at Section 4.2. Another consequence is that while clusters organize the information from the same geographical area, most of the queries will have a local effect on the cluster. This contributes in a system isolated from traffic from other parts of the network, as well as a *likely* greater speedup in the network communication.

Table 2.6: Evaluation of systems on spatial queries.

System	Topology	Dimen–sionality	Mapping approach	Storage Efficiency	Time Efficiency	Routing Load Bal.	Data Load Balancing	Correctness	Completeness	Intersection (side)
Spatial Query [67]	Hypercube (CAN [11])	2-D	1:1	$O(o_S \times \mathcal{O})$[12]	$O(\sqrt{N} + o_S \times \mathcal{O})$	Yes	No	Yes	Yes	System
Liu *et al.* [57]	Tree (Super-peers [56])	M-D	N/M	1	$\Psi(N, \lvert C \rvert, S)$[3]	No	N/A	Yes	Yes	System
DHR-Tree [68]	Tree (P-Tree [69])	M-D	N/M	1	$O(\log_d N + S \times N)^4$	Yes	N/A	Yes	Yes	System
Globase.KOM [20]	Tree (Super-peers)	2-D	M:1	1	$O(\log \lvert C \rvert + S \times C_N)^5$	No	N/A	Yes	Yes	System
SDS [66]	Tree (Quadtrees [70])	2-D	M:1	$O(o_S \times f_{min})^{16}$	$O(\log N + f_{max} - f_{min})^6$	No	No	No	Yes	N/A

[1] o_S specifies the ratio of the covered data domain for rectangular data objects.

[2] $\mathcal{O}$ refers to the application data domain.

[3] The cost depends on the number of nodes into the system N, the number of clusters $|C|$ and the query selectivity ratio S.

[4] d is the order of the R-Tree.

[5] $|C|$ is the number of clusters, and C_N refers to the number of nodes of the target cluster.

[6] f_{min} is the minimum level in the quadtree from where insertions are done. f_{max} is the maximum level of the quadtree.

2.8 Content Distribution Techniques

In this section we explore the most remarkable techniques for the content distribution in a distributed system. In particular, we analyse application-level multicast and publish/subscribe services in the large scale. Briefly speaking, there have been several attempts to provide multicast services at IP-level at the Internet scale. For instance, HDVMRP [71], HIP [72] or GMRP [73] are cases of scalable, hierarchical systems proportioning such functionality. Since they seek efficient, non-intrusive IP-level multicast services, these systems consider several levels in the hierarchy (e.g., GMRP [73]), so that each level is self-organized in the same fashion than Autonomous Systems do at Internet scale. For efficient communication, systems construct tree-like network structures, given that they fulfill naturally with the *one-to-many* communication model of the publish/subscribe paradigm. Despite all efforts, IP-level multicast is not ubiquitously deployed. This motivated that some systems (like [74]) try to complement the existing islands where IP-level multicast services are available, spreading over the gaps application-level multicast services, or just deploying new peer-to-peer publish/subscribe and application-level multicast services, regardless the underlying network.

Actually, IP-level multicast services are out of scope of our analysis. Our particular focus is put on *peer-to-peer application-level multicast and publish/subscribe services*. To do so, we motivate the necessity of such services by an example application in the following section. Afterwards, in Section 2.8.2 we introduce a general design for an application providing publish/subscribe services. We then introduce the definition of the types of publish/subscribe techniques in Section 2.8.3.

In Section 2.9 we introduce the parallel computing abilities of the publish/subscribe algorithms. We then elaborate on the parameters that we include in the evaluation framework for the publish/subscribe techniques in Section 2.10. We continue with the comparison study of systems (in Section 2.11) following either the topic-based model (Section 2.11.1), or the content-based model (Section 2.11.2).

2.8.1 Use Case: Publish/Subscribe Application

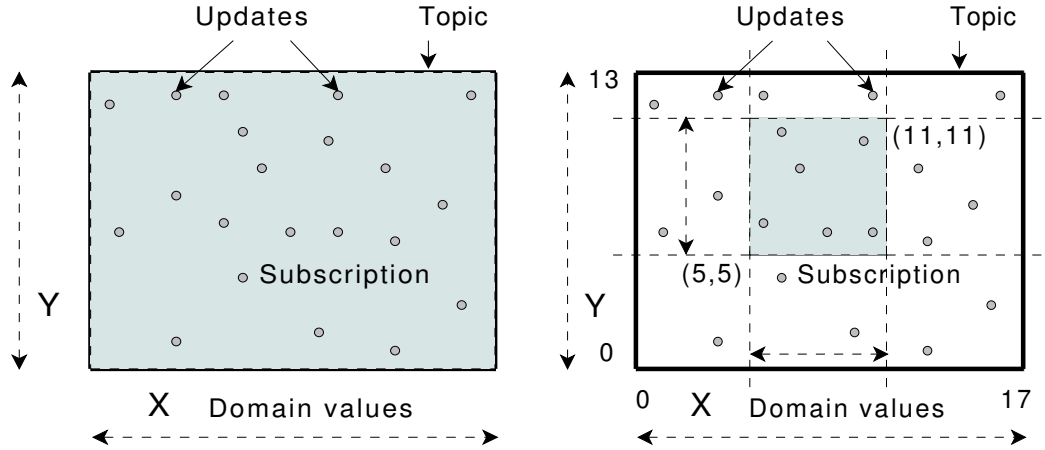
To let the reader understand the necessity and complexity of providing publish/subscribe services over distributed systems based on SPNs, we explain two applications where such a functionality is required, among scalability by service decentralization and high performance by operation parallelization. Internet has become a tool for

user collaboration and distribution of information, where such richer functionalities are demanded.

The application we consider is the *dissemination of pieces of information*, like news or interesting messages, for a large number of participants. The goal of this application is that (from several to thousands of) nodes should be able to send news (namely *events*) to all the system participants in an efficient and scalable way, which have *registered* their *interest* in that *kind of news*. This exemplifies a *many-to-many communication* pattern. That is, all participating nodes record their *interest* in the *subject* or *topic* (or in the *content*) of the disseminated news. Since IP multicast is not ubiquitously deployed across heterogeneous networks, like Internet, *application-level multicast* and *publish/subscribe* systems are used instead. *Topic-based* publish/subscribe or application-level multicast systems (namely *TOPS* systems) overcome the problem of *subscribing* users' interests to a certain *kind of news*, to wit *topic*, (like "sports" or "jazz music"). Likewise, *content-based* publish/subscribe systems (namely *COPS* systems) allow users to receive only those news that fulfill some *rules* imposed on their *content* (like {*TOPIC*="sports" *AND* (*CONTENT* **has** "Fernando Alonso" *OR CONTENT* **has** "F.C. Barcelona")}).

In particular, a subscription can be seen as a static range query, that continuously filters the received events into two classes (See Fig. 2.7). The useful events fall down into the subscribed area (shadowed area in the figure). Conversely, useless events fall apart from the user interests and are discarded.

This kind of applications presents some common elements like in the case of similarity queries. In both TOPS and COPS techniques, the (distributed) application necessitates some *algorithms* to disseminate new events through the nodes that are interested in them. In order to decide whether an event must be communicated to a certain participating node, these applications determine a *matching rule*. Whether the *matching rule* is accomplished, the event is notified to the given node. But the *matching rule* differs when either a TOPS or COPS technique is considered. In TOPS model, the matching rule is whether an event refers to a certain topic (see Fig. 2.7a). Instead, in COPS model, the matching rule analyses whether the *conditions* imposed on the content of potential events are fulfilled (see Fig. 2.7b). Broadly speaking, COPS applications consider a *multi-dimensional* data space, where subscriptions specify ranges of interest (i.e., the conditions) and events are datums within the given data space.



(a) Topic-based subscription example.

(b) Content-based subscription example.

Figure 2.7: Examples of subscriptions in a 2-D scenario. In both cases, the shadowed area represents the *covered* area for the given subscription. In (a), in the topic-based model, any event of the given topic is useful for the application. Instead, in (b), the content-based model allows applications to filter out the events according to its content. The subscription $s = \{[5..11], [5..11]\}$ will select only the events that match the shadowed area. The other ones will be useless for the application and, thus, they will be discarded.

2.8.2 System Design: An Overview

After having introduced an example of publish/subscribe system, we turn into the description of a general design of such kind of systems. We depict the generic design in Fig. 2.8. The whole system is organized by several or thousands of instances (or nodes), each of which is constituted by two layers: *Node* and *Application*. Each layer has its own functionality. The *Node* layer is responsible of routing, whenever necessary, new events towards other nodes that potentially are interested in them. Moreover, subscriptions (or alternatively advertisements) can be routed to some other node(s) in order to provide a more efficient organization of nodes within the network. Briefly speaking, in some solutions nodes uses *advertisements* to notify the system the kind of events that they will produce thereafter. The *Subscription Manager* entity is responsible of deciding whether to forward events, subscriptions and advertisements, or to store subscriptions and advertisements, or notify new events, into the *Application* layer. The *Data Indexing* store could be used to quicken these decisions, such as calculating if the node is responsible for a given message, by avoiding to access to the Application layer. When forwarding is necessary, the node employs the convenient *Routing Algorithms* and selects some node(s) from the set of *Neighbors* where the message (i.e., event, subscription or advertisement) should be forwarded to.

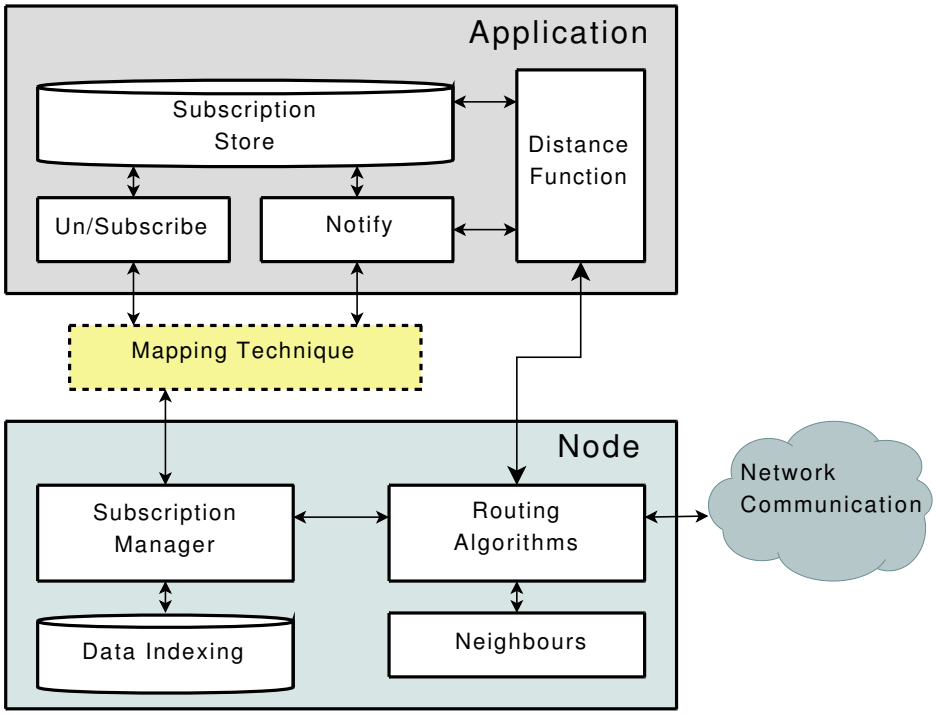


Figure 2.8: Generic layered design, common components and information flow of distributed publish/subscribe systems.

The *Application* layer addresses the local management of events and subscriptions instead. It *store*s the subscriptions for a later matching against events and, consequently, to start the *notification* process. When applicable, it also manages advertisements. The *distance function* helps on filtering out events and forwarding accordingly incoming messages, such as subscriptions targeting other nodes from the system. The *Mapping technique* is used in some solutions to adapt subscriptions and events (resp. advertisements) to the SPN keyspace, so that these messages can be effectively routed and managed by the network. The module *Un/Subscribe* updates the local subscription store with new subscriptions (resp. removing them when subscriptions are cancelled or when their leases are over). The counterpart module *Notify* carries out the matching process of incoming events against the local store of subscriptions. The result of this process is usually a list of nodes which should be notified to. To do so, the node uses the corresponding *Routing Algorithm* for distributed event notification. This module also notifies the local application when events lies on its interest.

2.8.3 Publish/Subscribe Services: Definitions

Publish/subscribe systems are powerful mechanisms for information dissemination in a distributed setting. These systems are characterized by two main actors. *Publishers* are those actors who produce information. Usually, the literature denotes such pieces of information as *events*. Additionally, *subscribers* are those actors that are interested in receiving significant events. They employ *subscriptions* to define their particular interests, expressing conditions on the content of events (*content-based* model) or just on a category they belong to (*topic-based* model). In some cases, *publishers* inform the system of the kind of events they will publish. To do so, publishers use *advertisements*. Let us now present the matching rule definition, as well as the kind of publish/subscribe systems.

Definition 2.7 (Matching rule) *Let $\mathcal{E}$ and $\mathcal{S}$ be the set of events and subscriptions, respectively. Let $\mathcal{B}$ be the set $\{true, false\}$. The function $\mathcal{M} : \mathcal{E} \times \mathcal{S} \rightarrow \mathcal{B}$ express whether an event matches a subscription.*

Given this matching rule definition, we provide the following statement for a general publish/subscribe service:

Definition 2.8 (Publish/subscribe service) *A publish/subscribe service is such that any event E from a publisher p is delivered to any subscriber s with subscription S, such that $\mathcal{M}(E, S)$ succeeds.*

This definition states a *many-to-many* communication pattern, where useful events must be delivered to interested subscribers, no matter which are the publishers and subscribers nodes. Therefore, nodes must collaborate in order to disseminate events to interested subscribers. To do so, nodes are organized in such a way that the event distribution becomes efficient in the large scale. Nevertheless, the model of the publish/subscribe system poses different challenges in the way the network organization is addressed. Let us introduce how events and subscriptions match in either model.

Definition 2.9 (Topic-based matching rule) *Let $topic(\Delta)$ denote the topic (i.e., category) that Δ belongs to. The matching rule in a TOPS system is as follows:*

$$\mathcal{M}_{TOPS}(E, S) = \begin{cases} true & topic(E) = topic(S) \\ false & otherwise. \end{cases}$$

In the literature, the topic-based model is also referred to as *application-level multicast* (because it allows multicasting events to different target nodes) or also as *channel-based model* (since every topic virtually constitutes a channel where information flows).

Provided the above definition, it is easy to see that any node that is interested in the topic will receive its events, regardless the content of the events (as it can be seen from Fig. 2.7a). That is, in TOPS systems, subscriptions and events take the form $S = \{topic\}$ and $E = \{topic, content\}$, respectively. The same is not true for COPS systems though: In the content-based model, the event's content matters.

Broadly speaking, one can see the data space for COPS systems as a *multi-dimensional* data space $\mathcal{O} = \{V_1, V_2, \ldots, V_m\}$, with $|\mathcal{O}| = m$ dimensions, where V_i is the data domain for the i-th dimension, $i = 1, \ldots, m$. Thus, an event E is a *point* into $\mathcal{O}$. Namely, $E = \{topic, \{E_1, \ldots, E_m\}\}$, having $E \in \mathcal{O}$ and $E_i \in \mathcal{V}_i$, $i = 1, \ldots, m$, where E_i is the event's content for the i-th dimension. Conversely, a subscription is a more complex data structure that specifies filter rules into one or more dimensions from the data space, with the form $S = \{topic, \{S_1, \ldots, S_k\}\}$, $k \leq m$, where S_i denotes a filter rule for the i-th dimension, selecting one or more values from data domain V_i. The following function introduces the percolation process for any individual filter rule applied to the event's content.

Definition 2.10 (Percolation function) *Let $\mathcal{R}$ be the set of filters for a data domain $\mathcal{V}$. The function $\rho : \mathcal{V} \times \mathcal{R} \rightarrow \mathcal{B}$ tells whether a given value $v \in \mathcal{V}$ passes the filter $r \in \mathcal{R}$.*

Once we have the percolation function, we have the necessary tools to introduce the matching rule for the content-based model. The idea behind this model is that *all filtering rules* should be passed by a given event in order to notify it to the corresponding subscriber.

Definition 2.11 (Content-based matching rule) *The matching rule in a COPS system is defined as follows:*

$$\mathcal{M}_{COPS}(E, S) = \begin{cases} true & topic(E) = topic(S) \wedge k \leq m, 1 \leq i \leq k : \forall i | \rho(E_i, S_i) = true \\ false & otherwise. \end{cases}$$

The above matching definition for the content-based model states that a more complex solution is necessary to deal with a COPS service. The reason is clear, since the same event E may not be of interest to all subscribers who matter the targeted topic, for instance.

The existing gap from subscripting and receiving notifications of related events is addressed in different ways in the distributed field. Namely, a publish/subscribe system necessitates a mechanism to meet an event with subscriptions, so that all interested nodes are notified with the new information. Covering [75], epidemic-based [22, 76], summarization [77, 78], source-based [79] and rendezvous [21, 80, 81, 82, 83] are

the common techniques used in peer-to-peer systems, each of which determines the necessary kind of node organization in order to work properly. Nevertheless, the most deployed technique in SPNs is the **rendezvous model**. The reason behind that is because the multi-hop routing abstraction implemented by SPNs integrates naturally with the need for globally unique rendezvous points (i.e., nodes). To this end, events and subscriptions are adapted to the SPN keyspace, so that they can be processed by the SPN routing algorithms. To illustrate how, let us introduce the adaptation function of this kind of complex information. For the sake of clarity, let us refer events and subscriptions as *complex objects*.

Definition 2.12 (Complex object adaptation) *The function* $\mathcal{F}_{\mathcal{CO}} : \mathcal{O} \rightarrow 2^{\mathcal{I}'}$, *where* $\mathcal{I}' = \mathcal{I} \setminus \emptyset$, *adapts a complex object from the data space* $\mathcal{O}$ *to a set of keys.*

Notice that the result of $\mathcal{F}_{\mathcal{CO}}$ will produce a set with at least one key $k \in \mathcal{I}$ (i.e., the empty set $\emptyset$ is removed at $\mathcal{I}'$). This function explain in a generic way how events and subscriptions are adapted to the SPN keyspace. For instance, $\mathcal{F}_{\mathcal{CO}}$ for a given event E constructs the set of keys ks_E where the event must be sent to (to wit $ks_E = \mathcal{F}_{\mathcal{CO}}(E)$). Conversely, $ks_S = \mathcal{F}_{\mathcal{CO}}(S)$ does the same for a subscription S. This way, the rendezvous nodes are naturally selected to be the responsible ones of the keys in ks_E and ks_S, and they will react as it was designed in the COPS system (e.g., storing the subscription S, or matching the event E against the locally stored subscriptions, in order to draw the nodes to notify to).

2.9 Parallel Computing on Publish/Subscribe Services

The aim of these systems is to deliver (i.e., in a push-based approach) all events from publishers to corresponding subscribers. To do so, some of the existing publish/subscribe systems build a specific publish/subscribe peer-to-peer overlay [22, 76, 77, 84], also called *event brokering networks*. Nevertheless, we focus on those solutions that leverage SPNs [7, 8, 9, 11], like Scribe [21], Bayeoux [79] or PastryStrings [83]. The idea behind that is to construct a *descentralized service*, deployable at the large scale, and benefiting from *operation parallelization*, since all participating nodes collaborate in the dissemination tasks. We ellaborate on the system properties that matter for the process parallelization in Section 2.10.3.

Regardless of the topic- or content-based model, they employ different subscription mechanisms in order to meet events and the corresponding subscribers. This contributes that nodes cooperating in the distributed publish/subscribe system have

guarantees for publishing and subscribing *at any time and concurrently*. In particular, as we have stated before, the *rendezvous model* is the most deployed into SPNs, since it is naturally supported by SPNs. In addition, in some cases like [78, 85], systems employ advertisements from publishers in order to meet both subscriptions and events. In some other cases [85, 86], the rendezvous model helps on meeting subscriptions and advertisements.

Since the service distribution is clearly stated by deploying it upon a peer-to-peer system, we delve into the parallelization of dissemination tasks in the following section. We detail some generic algorithms used to disseminate events among subscribers. Actually, this is the key component on publish/subscribe systems, since an efficient event distribution is expected and can be highly parallelized. Conversely, we do not focus on the subscription task, because it is performed by a singular node and takes little effect on the system. Nevertheless, existing solutions determine a specific subscription mechanism, so that further subscriptions do not shrink the event dissemination performance. Thus, the reader should expect very odd subscriptions techniques, but with a common goal, an efficient event publication.

2.9.1 Parallelizing Event Dissemination

For the sake of simplicity, let us consider that both publishers and subscribers participate in the network, so that all the communication is performed within the peer-to-peer system. Algorithm 2.4 depicts a default process for the event dissemination. Since the role of publisher and subscriber may not coincide at each node, the algorithm necessitates to check whether the visited node is subscriber for the event's topic (lines 2–4). Afterwards, the given *node* must forward the new event to other neighbors (lines 5–8), except to that where the event was received from (line 5).

One could think that this dissemination algorithm never ends. An assumption is taken here: Nodes are organized in such a way that no loops appear when distributing the topic's events. That is, the publish/subscribe overlay constitutes a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with no cycles, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ the set of edges to each other. Some systems develop a new overlay network over an existing SPN (such as Scribe [21]) or some other send events to one or more rendezvous nodes (like in [82]). This means that, either explicitly (by the constructed publish/subscribe overlay network) or implicitly (by the event dissemination algorithm), the existing SPN-based solutions develop a communication tree to address the problem of an efficient *one-to-many* communication pattern.

Algorithm 2.4 $notify$
Input: *node* /* *node where algorithm is executed* */
Input: *sender* /* *node who forwards the event* */
Input: E /* *event to disseminate* */

1: $S_{node} \leftarrow$ subscription of *node*
2: **if** $\mathcal{M}(E, S_{node})^a$ **then**
3: $\quad local\_notify(node, E)$
4: **end if**
5: $to\_visit \leftarrow neighbors(node, E) \setminus \{sender\}$
6: **for all** $neigh \in to\_visit$ **do** /* in parallel */
7: $\quad notify(neigh, node, E)$
8: **end for**

[a] The matching rule $\mathcal{M}$ becomes $\mathcal{M}_{TOPS}$ for TOPS systems or $\mathcal{M}_{COPS}$ for COPS systems.

Moreover, if we consider explicitly the *rendezvous model*, we obtain a two-phase problem solver algorithm, as shown in Alg. 2.5. Along the first phase, the event is forwarded against its rendezvous point (lines 1–8), the node which will be the responsible to match it with existing subscriptions. Once the rendezvous node is reached, the second phase starts (lines 9–11), which invokes the aforementioned $notify$ algorithm (Alg. 2.4).

Algorithm 2.5 $publish$ in rendezvous model
Input: *node* /* *node where algorithm is executed* */
Input: *sender* /* *node who forwards the event* */
Input: E /* *event to disseminate* */
Input: k /* *key in* $\mathcal{I}$ *from E's adaptation; default to* $\emptyset$ */

1: **if** $k = \emptyset$ **then** /* Starts routing phase */
2: $\quad$ **for all** $key \in \mathcal{F}_{\mathcal{CO}}(E)$ **do** /* in parallel */
3: $\quad\quad neigh \leftarrow best\_neighbor(node, key)$
4: $\quad\quad publish(neigh, node, E, key)$
5: $\quad$ **end for**
6: **else if** *node* is not responsible for k **then** /* Routing phase */
7: $\quad neigh \leftarrow best\_neighbor(node, k)$
8: $\quad publish(neigh, node, E, k)$
9: **else** /* Starts the notification phase */
10: $\quad notify(node, node, E)$
11: **end if**

Notice that Algorithm 2.5 reflects two different stages on the routing phase along

the publication process. The first stage (lines 1–5) is addressed by the first node (i.e., the publisher) starting the procedure. The specific adaptation technique ($\mathcal{F}_{\mathcal{CO}}$ at line 2) produces one or more keys which the event is directed to (lines 3–4). The second stage (lines 6–9) is performed at intermediate nodes, whose goal is to forward the event until the k's responsible node is reached (i.e., the rendezvous node).

2.10 Evaluation Criteria for Publish/Subscribe Services

The systems target of this study provide publish/subscribe services either in the topic-based or content-based model. As we presented for the systems providing similarity queries in Section 2.4, the current systems are analysed accordant with a set of qualitative and quantitative parameters, constituting a *common evaluation framework*. We characterize each parameter in Table 2.7. Even though some of them are the same than in the previous study, we replicate them here for the sake of reader's simplicity. By using this framework, we construct firstly the systems' portrayal, describing the key components and parameters from the solutions, what facilitates a fair comparison among systems. The parameters can be considered broadly of two types: (i) **implementation** parameters, that refer to the system design and construction, and (ii) **quality of service** parameters, that consider the quality of the solution on subscription management and event dissemination. The parameters' definition is stated in the following sections. An analysis of parallel computing abilities according to these properties is also included.

2.10.1 Implementation Criteria

To decide which parameters reckon into the evaluation framework, let us consider our application example of news dissemination. As we have seen in the Section 2.9 Parallel Computing on Publish/Subscribe Services, the portrayed algorithms have a key routing component. Thus, the node organization, to wit the **topology**, have an important effect on the performance of the dissemination algorithm. Indeed, it also will dictate where the information will be managed for global efficiency. This motivates the inclusion of the topology into the evaluation framework.

As seen in the data management evaluation criteria (Section 2.4), we also consider the **dimensionality** and the **mapping approach**. The dimensionality defines the number of attributes (or dimensions) that the application data domain consists of. Conversely, the mapping approach adapts the application data space to the SPN keyspace to work properly. Even though these properties matter obviously for content-based

approaches, topic-based solutions do not consider the content of events and subscriptions. Therefore, the dimensionality and mapping approach will receive almost no attention.

Table 2.7: Characterization of the evaluation criteria for publish/subscribe systems.

Criteria	Characterization
Mapping properties	
Topology	The **topology** defines a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where vertexes $\mathcal{V}$ are nodes, and edges $\mathcal{E}$ the links established between nodes. In addition, only edges $\mathcal{E}$ are used for node inter-communication in normal peer-to-peer operations. The topology can be broadly specified as *structured*, when the topology relies on any geometric form (e.g., ring, hypercube, tree), and *unstructured*, when nodes are connected in a non predefined fashion (e.g., a mesh).
Dimensionality	Systems that provide this kind of high-level services in a distributed way can support either one-dimensional (**1-D**) or multi-dimensional (**M-D**) application domains. When possible, systems will be analysed for both sorts of dimensionalities.
Mapping approach	Because in most cases application domains do not correspond to SPN keyspaces, the distributed application needs to adapt the information in order to allow its indexation by the SPN. This kind of data transformation is called also *mapping* and defines exactly the way the transformation is achieved. We can find these sorts of transformation: **1:1** (Left) One application dimension is mapped to a single SPN keyspace (right). **M:1** The whole multi-dimensional application domain is mapped to a value in the SPN keyspace. **M:P** The whole multi-dimensional application domain is transformed to a P-dimensional SPN keyspace. **N/M** When no mapping is applied.

Continued on next page

Table 2.7 – *Continued*

Criteria	Characterization
Subscription and notification properties	
Storage efficiency	In a distributed or parallel system, the amount of storage used to record information is significant to reduce notably the response time. In general, the more copies of the same information, the less delay on answers. As events are ephemeral information, we use this property to quantify the *overhead* that different algorithms pose on subscription storage.
Time efficiency	Under this property we measure the amount of time needed by both the distributed algorithm to perform a subscription, as well as the dissemination process. This property accounts the amount of time the algorithm takes for visiting the last node involved in the procedure, either for a subscription storage or event notification.
Load balancing	When operating in a distributed or parallel system, it is convenient that all nodes have (approximately) the same amount of *load*. The term *load balancing* here means both *data* and *routing load balancing*: the former property tells whether all nodes (approximately) manage the same amount of information (i.e., subscriptions); the latter depicts that the system is able to route throughout the set of nodes without hotspots, especially along the event dissemination process.
Noise	This property accounts a complementary measure to the *load balancing* for the critical process of event dissemination. The goal is the quantify the *overhead* put on the system, as the number of visited nodes apart from those strictly necessary. Actually, the best case for a dissemination process for an event E is just visiting the nodes interested in E (where the system incurs no *noise*).

2.10.2 Quality of Service Criteria

The goal of a publish/subscribe system is the efficient dissemination of events. To do so, systems deploy a specific *subscription management* which dramatically determines their efficiency. This way, from a system's viewpoint, events are ephemeral data that are not stored, but just distributed. However, publish/subscribe systems must deal with subscriptions from nodes, storing them somewhere in the system for global efficiency. Even though that a single copy of the subscriptions stored in the system would suffice, the truth is that both the SPN network structure and the application data domain can pose an obstacle, so that more complex solutions are demanded. This clearly

motivates the introduction of the **storage efficiency** into the evaluation framework. The stress put in this property also affects the capacity of nodes, and thus the scalability of the solution. We consider under this property neither replication nor caching techniques in this study. Actually, they are applied to a wide variety of distributed systems, what includes the systems analysed in this study.

Another important issue on the publish/subscribe services is the *completion time* of all the tasks. Publish/subscribe systems deal with subscriptions and events. Firstly, whether the subscription S is stored locally at subscriber p or it is stored in some other node q different from p, will reflect on the completion time for the subscription process. As we have depicted before, the idea behind the subscription storage and management is to provide an efficient event dissemination. Secondly, event distribution is the key component of publish/subscribe systems and, therefore, the time required to notify all interested nodes is crucial. Factors like the SPN structure or the subscription management policy settle the completion time of the event dissemination. We summarize all this into the **subscription** and **event time efficiency** properties. To do so, we measure the number of nodes necessary to visit to conclude either operation.

Inasmuch as all nodes cooperate in some degree to provide the publish/subscribe services, we also need to evaluate the fairness in their cooperation. We center our focus on the two most critical factors: storage and process. The former complements the above *storage efficiency* property, since it considers the efficiency from a system's viewpoint. Instead, we here consider the fairness in storage among individual nodes in the system. For instance, the subscription storage could be not consistent among all nodes. The latter serves as a complement to the *time efficiency*. Time efficiency considers the completion time from a system's viewpoint, while we here consider the time dedicated by any individual node to the completion of tasks. For example, in some cases the participation in the message passing along the routing algorithms could be unequal, overloading some nodes. We then summarize all this factors under the terms **data** and **routing load balancing**, with which existing inefficiencies come to light.

The last property we consider in the evaluation framework is the overhead, namely **noise**, in the event dissemination process. In particular, since events have to be delivered to all interested subscribers, the network structure could necessitate not only the targeted subscribers, but also some additional (routing) nodes to forward events to destination nodes. We summarize under the *noise* property the number of routing nodes visited by the dissemination algorithm.

2.10.3 Parallel Computing Evaluation

The reader can realise that most of the statements produced in the parallel computing evaluation for similarity queries (Section 2.4.3) can be also applicable in this scenario. In the following we detail the most important parameters that influences in the parallel computing on publish/subscribe systems. In particular, we focus on the event dissemination parallelization, since it is the process that carries most of the process load.

The consequence of the parallelization of the event dissemination in a distributed setting is twofold: the completion time is shorter, and the usage of system resources increases. In particular, given that SPNs perform node inter-communication by message passing, the bandwidth usage increases between all participating nodes, as well as the usage of nodes' resources (e.g., computing cycles and main memory). Given that SPNs provide an inherent way of parallelizing tasks, we provide in this section an analysis on the parameters from the evaluation framework, which determine the feasibility and efficiency of the parallel computing. Parameters appear detailed in order of significance.

The most important factor is the system's **topology**. The way that nodes are organized dictates if operations can be performed in parallel and how. For instance, we have depicted an algorithm to distribute events that is fully parallelizable (see Algorithms 2.4 and 2.5). Actually, the idea behind that is to disseminate events timely in the large scale, using a one-to-may communication model. As this hints, a tree-like communication pattern is mostly followed to help fostering a shorter completion time of the event distribution. This way, an inherent goal in all the publish/subscribe designs is to constitute a solution as much parallelizable as possible.

Nevertheless, not all systems develop the same kind of **algorithms** to (firstly) manage subscriptions, so that (secondly) the distribution of events becomes rapid and resource efficient. In fact, algorithms decide greatly the quality of the overall solution, as well as their scalability. In addition, the amount of **subscription storage** also influences the possibility to scale. For instance, let us suppose a fixed number of nodes N: the more subscription storage necessary is, the busier the nodes, so that the system can manage globally less subscriptions and, thus, poses a hindrance to scalability.

Another important issue to successfully parallelize an algorithm is the **load balancing** property. That is, preventing that some nodes become overloaded (i.e., hot spots), the system benefits for wider possibilities to effectively parallelize tasks. For instance, if a node p should store a great amount of subscriptions, p will be overloaded by the amount of managed subscriptions and by the reception of events. The former

provokes a *data load unbalancing*, given that the data management is not fair among nodes. The latter, also in consequence to the first point, makes p to turn a *routing hot spot*, because it must receive a greater amount of events and start the corresponding notification processes.

The last properties we consider are the **time efficiency** and the **noise**. Since the communication between nodes is conducted by message passing, there are two elements that also decides the parallelization performance. Firstly, the *delay* for node inter-communication is important to shorten the completion time. In addition, the less participating nodes, the faster the operation is concluded. This way, whether the given solution includes some helper nodes (i.e., *noise*) for the event dissemination, it also contributes on parallelization and shortening the completion time.

In conclusion, there are lots of variables that influences the quality of the publish/subscribe service as well as the possibility to scale by parallelization. Thus, the reader should expect to read about systems with widely different solutions.

2.10.4 Evaluation Criteria: Tuning and Terminology

All parameters introduced before are considered within the evaluation framework for all systems. Actually, publish/subscribe systems considered into this chapter were designed so that, given an application context, they provide a trade-off between all these parameters. For instance, a publish/subscribe service could incur in noise in order to provide higher possibilities to parallelize the event dissemination, as well as for balance the routing load among nodes. For the sake of reader's simplicity, we show in this section the whole set of parameters, even though most of them are the same than in the case of similarity queries.

Table 2.8: Tuning of the evaluation criteria for publish/subscribe systems.

Evaluation criteria	**Tuning**[1]
Topology	Descriptive name
Dimensionality	1-D | M-D
Mapping approach	1:1 | N:1 | M:P | N/M
Storage efficiency	$\frac{1}{\text{Num. data copies}}$
Time efficiency	$\frac{1}{\text{Num. overlay hops}}$
Load balancing	Yes | No
Noise	Yes | No

[1] The term **N/A** can appear elsewhere when the tuning is not applicable for a given system.

Table 2.8 shows the common measurements used along the analysis of systems providing publish/subscribe services, in order to value the considered systems. Given that in both *storage* and *time efficiency* measurements, 1 is the best case, the denominator is only shown on the comparison tables. For storage efficiency we refer to the number of subscription's copies are necessary to install to receive all events that matter to the given subscriptions (i.e., $\mathcal{M}(E, S)$ succeeds, where E is the event and S the subscription). Conversely, under the time efficiency we account the number of overlay hops last either the subscription process or the event distribution, to reach the last participating node. In addition, notice that in the case of the *noise*, we only determine whether a system suffers from it. The reason behind that is because it is difficult to calculate and, indeed, systems usually does not measure this property, but focus on their efficiency and short completion times. Instead, we elaborate on the risen noise in our module providing publish/subscribe services (see Chapter 5).

In addition, before starting with the system evaluations, let us detail the common terminology we will use in the following. As happened with the evaluation of systems providing similarity queries, every work uses its own nomenclature, but for the reader's clarity, we unify them as much as possible to a common and simpler naming. The reader can find it in Table 2.9. Here, the *selectivity ratio* expression refers to the ratio of the data space $\mathbb{O}$ that the subscription is selecting from. Notice that this only affects to COPS systems, since TOPS solutions inherently have a 100% selectivity ratio for events from a particular topic T. For example, let us suppose an arbitrary subscription S with a selectivity ratio of 50% on the application data space $\mathbb{O}$ or on the i-th attribute data domain $\mathcal{V}$. This means that S draws events living on a half of the application data space $\mathbb{O}$ or the attribute data domain $\mathcal{V}$, respectively. In the following sections, we delve into the comparison of different publish/subscribe distributed systems.

Table 2.9: Common terminology along the evaluation of publish/subscribe systems.

Term	Description
N	Number of nodes within the network
N_T	Number of nodes participating in the topic T
M	Number of attributes of the application data space
M_D	Number of attributes included in the complex object
S	Total subscription's selectivity ratio
S_i	Selectivity ratio of the i-th attribute

2.11 Supporting Publish/Subscribe Services

In this section we turn into the evaluation of systems providing publish/subscribe services. At a first glance, one could suppose that most of the system would employ a hierarchical network structure, since it naturally deals with multiplicity of paths which would foster the parallelization of event dissemination. Surprisingly, it is worth noting that the vast majority of systems supporting this kind of service leverage a flat network structure. The reason behind that is because hierarchical systems are usually weaker than a mesh or flat network structure when guaranteeing optimal performance in front of network changes (e.g., a node or a link fails). Actually, there are very few examples that use a hierarchical architecture (like [22]). Broadly speaking, as we will see in the following, the one-to-many communication model is implemented by either (i) the construction of a new overlay network (which may overlap totally or partially a SPN), or (ii) a specific overlay routing algorithm, leveraging the underlying networked system properties.

There is a lot of work done in the field of publish/subscribe services, also in the peer-to-peer paradigm. Nevertheless, for the sake of conciseness, we only consider the most noteworthy systems under evaluation. We go on the current analysis by introducing the comparison of the most remarkable peer-to-peer TOPS systems in the following section, and the peer-to-peer COPS systems in Section 2.11.2. Note that all systems appear at the end of each section in a table, ordered by the year of publication, in order to reflect somehow the evolution on the present field.

2.11.1 Topic-based Publish/Subscribe Services

We consider into the analysis the following TOPS systems: CAN Multicast [87], Bayeux [79], Scribe [21] and Tera [88]. To help in this analysis, the reader can find in Table 2.10 a summary with all the properties considered in the evaluation framework.

First of all, notice that since here we analyse TOPS systems, the data space dimensionality does not matter (see Definition 2.9 for the event matching rule in TOPS systems). Apart from it, a common characteristic of all included systems is that they develop a new overlay network where to manage the publish/subscribe service. In particular, except in Tera, all these new overlays are built atop of an already existing SPN. Tera, instead, builds a randomized graph (i.e., a mesh) based on Cyclon [89]. Even though we focus on SPNs, we decided to consider Tera under analysis for its similar performance to existing SPN-based TOPS systems. To further analyse these

systems, we delve into the two main tasks addressed by a publish/subscribe service: subscription management and event dissemination.

The **subscription mechanism** employed by all these systems is very similar. Subscriber nodes pack the subscription into a message and send it to some neighbor. Actually, these systems reckon subscription messages as the tool to join the *publish/subscribe overlay* which manages the target topic. This way, a node is involved in several overlays. The underlying peer-to-peer network (namely *global network*) is used as a directory service, that helps to find and join the specific publish/subscribe overlay. This comes validated by the subscription time efficiency shown at Table 2.10, since it always considers N, the total number of nodes in the system. Complementary, a node participates also in as many publish/subscribe overlays as topics it is interested in. In particular, CAN Multicast and Tera builds sub-overlays of the same kind than the global network (CAN [11] and Cyclon [89], respectively), with only interested subscribers participating in (i.e., there is no noise). Conversely, Bayeux and Scribe build publish/subscribe tree-like overlays, over ring-based SPNs (Tapestry [9] and Pastry [8], respectively). To do so, they require that some intermediate nodes to collaborate in the process of the tree construction.

Additionally, it is easy to observe that this subscription approach suffers from high signaling traffic (e.g., for node or link failures), since not only must the global network self-adjust its network connections, but also potentially will the rest of publish/subscribe overlays. Finally, notice that we have accounted the storage cost for any subscription as unitary, considering the fact of joining the publish/subscribe overlay as a whole.

The above subscription technique aim at boosting the performance of the **event dissemination** process. And here is where we will see the major set of differences. Bayeux and Scribe reproduce the two-phase algorithm described at Section 2.9.1, composed firstly by the *publish* Algorithm 2.5 and afterwards the *notify* Algorithm 2.4. The reason behind that is because the event has to pass through the tree-like publish/subscribe overlay, from parent to children nodes. Consequently, the event (i) must reach the rendezvous node (namely *root*) for the given topic (i.e., executing the Alg. 2.5), where (ii) the notification to all interested nodes starts (i.e., running the Alg. 2.4) by distributing the event from the root to all its children, and every child to its children, and continuing similarly thereafter. It is clear than this scheme suffers from routing load unbalancing, given that all events have to traverse the root node of the publish/subscribe overlay. Instead, CAN Multicast and Tera only applies the *notify* Algorithm 2.4 to disseminate the event from the very beginning. This

comes motivated because these systems utilize efficient routing-based or gossip-based algorithms, respectively, to broadcast events throughout the corresponding publish/subscribe overlays. This approach is beneficial provided that it does not imposes routing hot spots.

It is worthy to see an additional system, the work of Jia Weijia *et al.* [90], even though it is not included in the table due to its similarity with CAN Multicast. The solution is a bit more sophisticated than the CAN Multicast. The key incentive is to reduce (from the system's viewpoint) the total completion time for the event distribution, considering network latencies. To do so, [90] combines a tree-based publish/subscribe overlay within the CAN network. More specifically, [90] builds a logical hierarchy of CAN areas, where an area's node is selected as the representative one. Representative nodes are drawn in a way so that all they constitute virtually a hierarchy of nodes with minimal latencies among them. When disseminating the event, [90] applies the two-phase process where (i) the event is forwarded to next representative nodes, and (ii) each representative node notifies the event to all nodes in its area, as well as to next representative nodes. In consequence, this solution takes the best from both approaches: (i) it benefits from a lower-latency communication model because of the tree-like network structure, and (ii) without suffering from routing hot spots.

Table 2.10: Evaluation of topic-based publish/subscribe systems.

System	Topology	Dimen–sionality	Mapping approach	Storage Efficiency	Subs. Time Efficiency	Event Time Efficiency	Routing Load Bal.	Data Load Balancing	Noise
CAN Multicast [87]	Torus (CAN [11])	N/A	N/M	1	$O(P\sqrt[P]{N})$	$O(P\sqrt[P]{N_T})$	Yes	Yes	No
Bayeux [79]	Tree (Tapestry [9])	N/A	N/M	1	$O(\log N)$	$O(\log N_T)$	No	Yes	Yes
Scribe [21]	Tree (Pastry [8])	N/A	N/M	1	$O(\log N)$	$O(\log N_T)$	No	Yes	Yes
Tera [88]	Random graph (Cyclon [89])	N/A	N/M	1	$O(\log N)$	$O(\log N_T)$	Yes	Yes	No

[1] P is the number of dimensions on the CAN system.

2.11.2 Content-based Publish/Subscribe Services

In this section we analyse the following COPS systems. Hermes [91], Meghdoot [81], CBOVER [80], CBP2P [82] and PastryStrings [83] leverage SPNs, whilst Sub-2-Sub [76] and DPS [22] are fully self-organized peer-to-peer solutions. This way, we elaborate on both kinds of approaches, covering a wider field in the peer-to-peer publish/subscribe paradigm. The reader can see the complete results of this analysis in Table 2.11. Note that Hermes is a unique, interesting solution that develops a solution from a software engineering viewpoint. Both events and subscriptions are translated to objects in an object oriented language programming. In consequence, the classes and subclasses of subscriptions define how the links into the publish/subscribe overlay are set, where events will flow through to get to interested subscribers. As in the section before, we illustrate the main properties of these systems delving into the subscription mechanism and the event dissemination process.

From the **subscription technique**'s viewpoint, we can differ from two big sets. The first set includes those systems where the subscription process is translated to **join a** particular publish/subscribe overlay, which embraces Hermes, Meghdoot, PastryStrings, Sub-2-Sub and DPS. When joining a publish/subscribe overlay, two scenarios are considered. Sub-2-Sub and DPS construct a unique overlay where any node participates in. The node's location within the network obeys to particular routing algorithms. Instead, the rest of the above systems use an underlying SPN as a directory service (like in the case of TOPS systems). To complete the subscription, systems apply no transformation to the data space, but in some cases, for a M-dimensional data space, a subscription is completed after M operations (like in the case of PastryStrings, which also increments its complexity). Therefore, except PastryStrings and Hermes, these systems have a unitary cost for subscription storage (accounting the unique joining process as a whole). Notice that Hermes considers the object oriented language programming as the data space, where from Hermes' viewpoint there is no transformation of the data space. However, it is easy to see that any existing application domain must be transformed to a class in the given object oriented programming paradigm. In addition, the subscription is completed under the expected communication time for any operation into the target system.

The second group of the subscription techniques includes CBOVER and CBP2P that **leverage** the routing capacities of **an underlying SPN**. That is, instead of building a new overlay, they use the SPN as a *directory service where to store subscriptions*. To do so, they tightly rely on the *rendezvous model* to match events against subscriptions. The idea behind this approach is that a subscription S have to be stored in all those

nodes where an event E matching S (i.e., $\mathcal{M}(E, S)$ succeeds) can be directed to. This is demonstrated by the increment on their storage requirements. In consequence, these systems are the unique that need to adapt the data space $\mathcal{O}$ to the target SPN keyspace $\mathcal{I}$. The completion time also reflects a last factor $O(NS)$ that represents the amount of nodes where a copy of the subscription should be stored.

When considering the **event dissemination** costs, the reader can observe that systems mostly demonstrate the network diameter as the primary factor on the event dissemination cost. Notice that nothing is shown in PastryStrings, whose event distribution process is not reckoned, but only the process of collecting the set of nodes to who notify the event.

Here the task parallelization takes a principal role, though. For instance, even though that CBOVER [80] expects a worst case $O((M + \alpha) \log N)$ cost, we could consider $M + \alpha$ as a constant by parallelizing tasks, so that its cost would be $O(\log N)$ with a little hidden constant. However, $M + \alpha$ remains as the factor of increment on the bandwidth usage, since for an event E of M dimensions, the dissemination process should contact firstly with M rendezvous nodes, and, after matching E against locally stored subscriptions, all rendezvous nodes would start the notification process to all α subscribers interested in E. The dissemination scheme is really different in DPS, instead. Since nodes appear in the network clustered by interests, the event dissemination algorithm firstly deals with discovering the cluster C of nodes interested in the event E, and afterwards a distribution algorithm is responsible for broadcasting E among all C's nodes. The Sub-2-Sub approach is very similar, even though it only considers that just cluster participants can emit events related to each other interests.

The efficiency of the event dissemination algorithms is notorious, given that all systems with much or less success can balance the routing cost. Nonetheless, Hermes, Meghdoot and PastryStrings always need to contact rendezvous nodes to start the notification process, so that they turn into hot spots. The price of such an efficiency is the noise. That is, all systems (except Sub-2-Sub) necessitate of intermediate nodes to help on forwarding events to rendezvous nodes or target cluster of nodes.

Table 2.11: Evaluation of content-based publish/subscribe systems.

System	Topology	Dimensionality	Mapping approach	Storage Efficiency	Subs. Time Efficiency	Event Time Efficiency	Routing Load Bal.	Data Load Balancing	Noise
Publish/subscribe systems built atop SPNs									
Hermes [91]	Tree (Pastry [8])	M-D	N/M	$O(\log N)$[1]	$O(\log N)$	$O(\log N)$	No	Yes	Yes
Meghdoot [81][2]	Torus (CAN [11])	M-D	N/M	1	$O(P\sqrt[P]{N})$	$O(P\sqrt[P]{N})$	No	No	Yes
CBOVER [80]	Ring (Chord [7])	M-D	M:1	$\Sigma_{i=1}^{M} NS_i$	$O(\log N + NS)$	$O((M+\alpha)\log N)$[3]	Yes	Yes	Yes
CBP2P [82][4]	Ring (Chord)	M-D	M:1	NS	$O(\log N + NS)$	$O(\log N)$	Yes	Yes	Yes
PastryStrings [83][6]	Tree (Pastry [8])	M-D	N/M	$O(M)$ $O(M(\log \lvert S \rvert))$	$O(M(\log N))$ $O(M(\log \lvert S \rvert)(\log N))$	N/A	No	Yes	Yes
Publish/subscribe systems not atop SPNs.									
Sub-2-sub [76]	Ring	M-D	N/M	$1 + \Delta$[5]	$O(\log N)$	$O(\log N_T)$	Yes	Yes	No
DPS [22][7]	Tree	M-D	N/M	1 1	$O(hN_T)$ $O(kN_Tk'h)$	$O(MhN_T)$ $O(MkN_Tk'h)$	No Yes	Yes	Yes

[1] Storage amount for either subscriptions and advertisements.

[2] P is the number of dimensions on the CAN system.

[3] α refers to the number of subscribers that are necessary to notify.

[4] The values shown correspond to the *key-space split* scheme.

[5] Under Δ we account the amount of memory used to store gossiping information.

[6] First row evaluates the costs for string data types, whilst numbers in the second one. $|S|$ is the mean length of the range of values that a subscription S selects.

[7] The first row depicts the costs of the *leader-based* communication scheme, whilst *gossip-based* is the second one. k and k' are the number of nodes visited in the first round and the second one in the gossip-based algorithm, while h is the height of the tree.

2.11.3 Publish/Subscribe Services: Conclusions

The desirable properties of publish/subscribe systems should be, among others, minimum signaling traffic, optimized bandwidth use and unaffected by the network dynamics. Nevertheless, for an efficient publish/subscribe service and given a particular application scenario, the works presented in this section assume some trade-offs among all the expected properties.

For instance, systems that construct new overlays overlapping (partially) existing ones have duplicated costs when dealing with network dynamics, like signaling traffic when nodes or links fail. Conversely, these approaches become more effective when disseminating events, since they must just *forward* events according to the publish/subscribe overlay (instead of routing them into the global network).

On the other way around, systems that leverage existing SPNs have the advantage that signaling traffic is reduced to the minimum amount. However, this kind of solution has slightly higher costs when distributing events, since it has to *route* events, passing through nodes that probably are not interested in them.

To conclude this section, we have been able to see that all systems without exception construct such an publish/subscribe overlay or system design so that the event dissemination can be fully parallelized, following the one-to-many communication pattern, from the publisher to all interested subscribers. In addition, as expected, the key difference between TOPS and COPS solutions is that the system design in COPS services is really more complex than in TOPS approaches, since they have only to guarantee a *channel-based* communication pattern, regardless of the event's content. For example, we have detailed that in PastryStrings, M rendezvous nodes should be contacted for an M-dimensional data space, before starting the event notification process.

2.12 Open Issues on High-level Services in Peer-to-Peer Systems

Before concluding this chapter, we want to discuss about what we consider the most important open issues in the field of high-level services in peer-to-peer-enabled distributed systems. In particular, we introduce the existing trade-off between data locality and load balancing. The transverse problem of the high dimensional data domains is also stated. And finally, we consider the study of peer-to-peer-based data networks as the hot topic for next practical and broadly deployable distributed large scale applications.

Data Locality vs Load Balancing

In dynamic environments, where users perform lots of data operations (insertions, deletions and lookups) as well as where users remain connected only for a certain period of time, fair data distribution in peer-to-peer systems is challenging. Indeed, the problem in the context of this chapter is twofold. Firstly, data objects should be placed with some data locality guarantee, so that similarity queries turn efficient. And secondly, at the same time nodes should afford approximately the same amount of data load. This way, note that not only nodes could be overloaded with too many data objects, but also with lookup processing with high probability.

We can classify the SPNs into two big sets. One set formed by systems that require of mapping techniques to support data object indexing, so as to convert and adjust application data domains to the SPN keyspace. The other set is formed by systems that operate with the specific application data domain. Traditionally SPNs make use of uniform hash functions to map and distribute information uniformly at random between nodes for load balancing. The mapping drawback is the *loss of data locality* though. To guarantee a certain data locality, order-preserving or locality-preserving hash functions appeared. For instance, space filling curves (SFCs) like Z-curve [40] or Hilbert Curves [35] embody the set of locality-preserving hash functions.

Provided that one of the goals of this thesis is to provide a SPN-generic solution, we do believe that systems that embrace explicitly the application data domain turn inflexible and tied to the target application. Therefore, we will further analyse on the field of hash functions in order to provide portable solutions, deployable on to most of the existing SPN systems.

High-dimensional Data Domains

It has been demonstrated that the concept of proximity in high dimensional space may not be very meaningful [92]. These results show that for certain classes of commonly used similarity functions such as the L_p-norm in the Euclidean space, the nearest and furthest neighbor are of the same relative distance to the query point for large classes of data distributions. The lack of relative contrast in terms of similarity is somewhat undesirable, since it is not clear whether the nearest neighbor is meaningful under such circumstances. Thus, the distance function becomes unstable. Some direct consequences of this instability is the inclusion of far away data objects during the query resolution. This overloads nodes in time consumption and usage of computer resource because of filtering far data objects, producing greater response times on data lookups.

Even though this problem is not properly from SPNs, these systems experience inefficiency when high-dimensional data domains are supported. This motivates the adoption of advances in the similarity calculation from other data management fields, like database systems. To put an example of SPN permeability, we have seen in this chapter some SPNs that were based on tree data structures, like binary trees, B+-trees and quadtrees, for node and distributed information organization. It will not be strange to see new SPNs based on last results from the similarity field in high dimensional domains, like algorithms for data clustering [93] or techniques for dimensionality reduction [94]. In particular, we focus on supporting high-dimensional data domains, while services performance remain efficient.

Peer-to-Peer-based Data Networks

Instead of inserting data objects into the SPNs, Peer-to-Peer-based Data Networks (PDN) pose the stress on distributed information indexing while data objects remain locally stored into nodes. This is the way like most of well-known file-sharing peer-to-peer-based applications work actually, like eMule. Thus, the indexed records in essence consist of a pair $index = \{object, address\}$, where *object* is a reference to the data object being indexed, probably with additional meta-data to help in search operations, and *address* refers to the node's address storing the *object*. This indexing structure divides search algorithms into two steps: 1) querying peers perform the lookup operation to the distributed indexing data structure, obtaining a set of *index*es from the system, and 2) querying nodes retrieve the related data objects from nodes whose *address*es appear in the result set.

PDN will remain the most deployed application in the future, because its design facilitates users controlling the content stored locally. The price, though, is information indexing in the user's computer, which is usually a very little and reasonable cost compared to the volume of the data objects. Indeed, we foresee that PDN will become the hot topic for practical and broadly deployable applications. The research focus will then be placed in efficient distributed indexing techniques. In addition, not only should the indexation technique be time- and storage-efficient, but also it should help in improving the efficiency of data object transmissions. This could be achieved providing indexes of data objects that live in the surroundings of the querying node, as PIRD does. The main shortcoming of PIRD is the strict network structure and number of nodes into the system. Actually, whenever PDN designs resemble to the Internet topology, or whenever the system takes advantage of the network infrastructure [95, 96], systems will then benefit from node proximity. Proximity can be then

measured in any valuable metric, like latency and/or geographical distance. This is also another key issue considered in the work of this thesis, where we believe than intelligent node and data organization are possible, in order to feedback the system with enhanced functionalities.

2.13 Summary

This chapter has presented an evaluation of different systems providing similarity queries and publish/subscribe services for distributed, structured peer-to-peer systems. The evaluation is addressed to systems providing range, k-nearest neighbors and spatial queries for similarity queries, and to topic- and content-based models for publish/subscribe systems. Each part started with the motivation of the necessity of the relative kind of services. A set of generic algorithms were introduced, that provide the main steps of the considered operations, emphasizing on the parallelization of tasks. Then, we provided the definition of the evaluation framework.

After that, a total amount of 31 different systems of the last 7 years are considered within this study. Thus, our study provides a wide overview of this kind of systems since the introduction of first structured peer-to-peer systems in 2001. In particular, this study focuses on the time efficiency of the given operations. To do so, a total amount of 10 properties for every system are collected from the specific works and discussed along this chapter. When necessary, apart from those 10 properties, additional terms are considered, like query parallelization or amount of node status, in order to provide a big picture of systems, and to guarantee a fair comparison.

Similarity query systems evaluations

Broadly speaking, peer-to-peer-based systems that provide similarity queries provide different search resolution performances according to three factors: the topology, the query resolution approach and the query type. In addition, they are not independent variables, but the topology greatly dictates the rest of the parameters .

Topology. In last years more and more systems aim at providing enhanced lookup services by means of hierarchical SPNs, as it is seen along the chapter. This is motivated from the fact that most of the hierarchical designs provide inherently parallel query resolution. Instead, most of the flat architectures provide a two-phase search algorithm, where the system routes the search as a single message until a first responsible node is reached, and thereafter the query is parallelized among all responsible nodes.

Query resolution approach. SPNs adopt two main query resolution approaches, which range from DHTs that were redesigned to support similarity queries, to distributed data systems that structurally provide them. In general, we can see that for the first case, systems employ order or locality preserving hash functions to map data objects to the SPN keyspace. For the last case, most of the systems index data objects adopting the original data domain without transformation.

Query type. The type of query also determines the reachable system performance. As we have seen in this chapter, region-based range and spatial queries are more parallelizable than k-NN and sphere-based range queries. Moreover, most of the spatial queries are constructed over hierarchical distributed systems, what nicely fits the spatial query resolution algorithm. Thus, the inherent parallelization of tasks shorten the response time, but this is not for free. It affords more bandwidth usage and busy nodes per unit time. Nevertheless, given that nowadays Internet connection bandwidth and computer resources are growing rapidly, bandwidth and computation power are considered as not so restrictive as some years ago.

Publish/subscribe systems evaluation

Conversely, peer-to-peer-based systems providing publish/subscribe services depends greatly on the subscription mechanism, which will dictate how events are disseminated through the network.

Subscription mechanism. We have seen two main classes of subscription techniques: joining a *publish/subscribe overlay* or using a *publish/subscribe service* (which leverages a SPN as a directory service where to manage subscriptions). The *overlays providing publish/subscribe services* are categorized into two topologies. As before, the topology is one of the major factors that decides how subscriptions are managed. Most of the solutions adopt a tree-like network structure, which benefits fostering naturally the event dissemination. The mesh is the other common kind of network structure where gossip-based algorithms maintain the network connectivity and construct interest-based clusters, for a rapid and localized event distribution.

Nevertheless, building new overlays is not always feasible, and potentially less efficient when they are develop atop other peer-to-peer networks. They incur in *duplicated signaling traffic* for the maintenance of all overlays where a node participates in, which *should be avoided*. Instead, *publish/subscribe services* are more capable to deal with network dynamics, since they leverage the underlying peer-to-peer network for managing event matching and notification, as well as subscription storage.

Event dissemination. In *publish/subscribe overlays*, events are forwarded along the network to reach all the interested subscribers. Therefore, this scheme is very efficient, since the publish/subscribe overlay dictates the dissemination path, but at the cost of the maintenance of multiple publish/subscribe overlays (e.g., one for each topic that a node is interested in, under the TOPS model).

Conversely, in *publish/subscribe services*,the dissemination of events is addressed by leveraging an existing peer-to-peer infrastructure. In this case, the solutions suffer from a slightly higher communication cost, because events are actually routed. However, the great advantage is that they have to maintain just a single overlay, greatly reducing the signalling traffic and becoming more reactive to changes.

Challenges addressed in this thesis

To sum up, the fields of high-level queries and publish/subscribe services require of more research in the context of scalable, distributed, peer-to-peer solutions. Some of the open issues are described in this chapter. We mainly aim at researching new algorithms that will lean to a generic design of high-level services. In other words, we focus our research effort on **investigating in the feasibility of constructing SPN-generic services**. The idea behind that is the development of a common infrastructure where distributed operations are performed efficiently, no matter which SPN is supporting such an infrastructure.

More concretely, we will have to tackle with the high dimensionality problem. In addition, our proposed solution will have to provide a good data load balancing among nodes. Conversely, our proposal will have to retain a certain data locality, too. Specifically, the data locality will boost complex operations (such as range-based queries) and will improve their performance, compared to significant existing solutions. To prove that such infrastructure (or framework) is feasible and that all the deployed operations are to be efficient, both in the number of nodes and in the number of dimensions, we will construct the necessary distributed algorithms to build three odd services.

In particular, we develop services providing range queries (in Section 4.1), spatial queries (in Section 4.2) and content-based publish/subscribe services (in Chapter 5), so that all they can coexist under the same umbrella, whilst performing efficiently. A key point in the design of our algorithms will be to take advantage of the peer-to-peer-based parallel computing capabilities, in order to boost the feasibility and effectiveness of our solution. To start with, we elaborate on settling down the basis of our framework in the next Chapter 3.

3

A Framework for Developing Application-level Services in Structured Peer-to-Peer Networks

In this chapter we state the general structure of our **framework**, providing a SPN-generic infrastructure where high-level services can be deployed easily. This chapter also presents the elements of our **data adaptation module**, key component in the overall structure.

3.1 Introduction

We have illustrated some generic designs of systems providing either similarity queries or publish/subscribe services, which, for the sake of simplicity, we re-illustrate in Fig. 3.1a and Fig. 3.1b, respectively. These designs presented a two-layer abstraction. The *Node layer* featured the SPN state, routing capabilities and preliminary message processing. Conversely, the upper *Application layer* mainly provided data storage, as well as the business logic and the distance function tool. All this was set up in a distributed environment, where several or millions of instances of nodes were deployed in the network. But, what should be the global structure where our framework would be involved? To explain, we introduce an overview of the factors engaged in the decision. See Fig. 3.2 for the overall design.

The aim of this work is (i) to provide a portable framework, (ii) which should support a multiplicity of high-level services. Should we reach such a framework design, we will construct a generic infrastructure: firstly, SPN-generic because the framework could be deployed onto (most of) the SPNs; and secondly, generic for applications, that require a distributed substrate to benefit from computation resources on edges and, consequently, scalability.

To do so, we move from the above two-layer design towards a three-layer infrastructure (see Fig. 3.2): application layer on top, our framework in the middle, and the

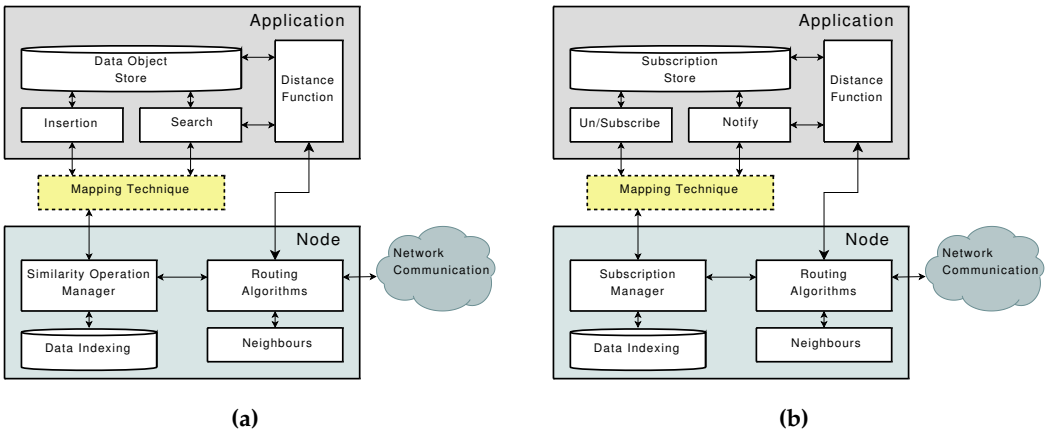


Figure 3.1: Generic layered design, common components and information flow of distributed systems providing (a) similarity abstractions, and (b) publish/subscribe services.

SPN layer on the bottom. This change, visually simple but extremely powerful, allow us to build a framework that will deal with all the presented challenges. Let us remember them briefly to put the reader in context:

Support of multiple applications. There are several concerns to consider from the applications, but in particular the following ones: (i) their data domains, (ii) their distance functions, (iii) their data storage, and (iv) the services they require.

As we have seen in the related work, applications usually are deployed directly onto an SPN, where applications have to deal with the services implementation (according to the underlying SPN), as well as to store the data objects locally when necessary. In addition, for the service working properly, they need a distance function tightly related to the application data domain, which tells how close are any two data objects. Clearly, the data domain, the corresponding distance function and the data storage should rely on the implementation of such applications. This is benefiting for the whole solution since this stands up a separation from application specific concerns to common concerns to several applications.

For instance, several applications could necessitate range query services or publish/subscribe services. Therefore, we move the services implementation into our framework layer, so that any application requiring the same kind of service could leverage our framework. More importantly, applications will not be involved implementing services that already exist into our framework.

Portable among SPNs. The idea behind that is to construct such a framework that can be deployed into most of the SPNs, taking advantage from their performance, efficiency and scalability, whilst supporting several services which should also render efficient and scalable. However, there are some other concerns to reflect in the framework design due to **SPNs profile**. We have already addressed a characterization of the SPNs target of this thesis in Section 2.1.1. In summary, the SPNs we will rely on, have a one-dimensional keyspace, where nodes have a set of neighbors which they are connected to. This connectivity guarantees that the communication between any two nodes takes a logarithmic number of hops.

Given that our framework leverage the SPN infrastructure as a communication framework, services included into our framework will be able to be reused among SPNs. In consequence, this makes cheaper the development of new services into our framework, since one service development can be reused by all applications that need such service.

Integrating applications and SPNs. The key issue of our framework, then, is to enable nicely the co-existence of both applications and SPNs into the same logical unit. In particular, note that the SPN keyspace is one-dimensional, while the application data domains are commonly characterized as complex, multi-dimensional data structures. For instance, an image can be characterized as a M-dimensional vector of M describing features.

Therefore, the integration problem is transformed to a **data domain management problem**. That is, we need to determine how application data domains and SPN keyspaces are both supported by our framework.

In order to support the application data domains in our framework, and consequently in the underlying SPN, we could address their management in two opposed fashions: (i) supporting directly the application data domain throughout our framework, and also in the underlying SPN, or (ii) adapting the application data domain to an application-uniform data space.

The former has the advantage that no data transformation is required to the application data domain, accelerating the solution by requiring no pre-process in the application data domain. However, we should set up as many different instances of our framework as the number of different application data domains.

Alternatively, the second option has the advantage that any application data domain is translated to a common data space, so that a single instance of our framework would suffice to operate with several applications and their data domains. However,

the main drawback is that this approach necessitates a pre-process to adapt the application data domains to a common data space. Since our goal is to provide a portable solution, we adopt the second approach in the heart of our framework.

In addition, as we have announced in Section 2.1.1, we reckon that SPNs have a uni-dimensional keyspace $\mathbb{J}$. Consequently, the target common data space to which transform any application data domain is going to be $\mathbb{J}$. This data domain transformation allows the framework to be instantiated over any SPN easily. To do so, we design a **data adaptation module** which factors the framework by concentrating and automatizing all data transformations from the application (potentially) multi-dimensional data domain to the SPN keyspace $\mathbb{J}$.

Nonetheless, there is an important challenge inherent to the data transformation technique. The adaptation technique should guarantee a necessary **data load balancing**, while, at the same time, place similar data relatively **close** after the adaptation. This is a major requisite to enable our framework render efficiently. Otherwise, distributed operations would become inefficient in terms of communication cost, or some nodes could be overloaded in terms of data storage.

Moreover, lots of high-level services, and in particular the services provided in this work (range queries, spatial queries and publish/subscribe services) have range-based operations as a key component. Since these procedures have to be deployed in a distributed environment, we also design a generic range-based algorithm to help on the distributed data management of complex range-based data objects (such as multi-dimensional range queries or range objects). As we can see from Fig. 3.2, these services are to provide some complex functionality to end-user applications.

The rest of the chapter is organized as follows. We describe the framework and its module structure in Section 3.2. We then dive into the *data adaptation module* at Section 3.3. In particular, we tackle how multi-dimensional data domains are adapted to the SPN keyspace in Section 3.3.1, and Section 3.3.2 addresses our proposed algorithm for range-based operations. We close this chapter with the concluding remarks at Section 3.4.

3.2 Framework Overview

Our framework is motivated for the expected genericity with the underlying SPN, as well as for the kind of featuring services. To this end, we have designed a three-layer scenario. To illustrate, see the Fig. 3.2. From a top-down reading, we firstly observe the application layer, our framework in the middle layer, and the SPN layer in the bottom

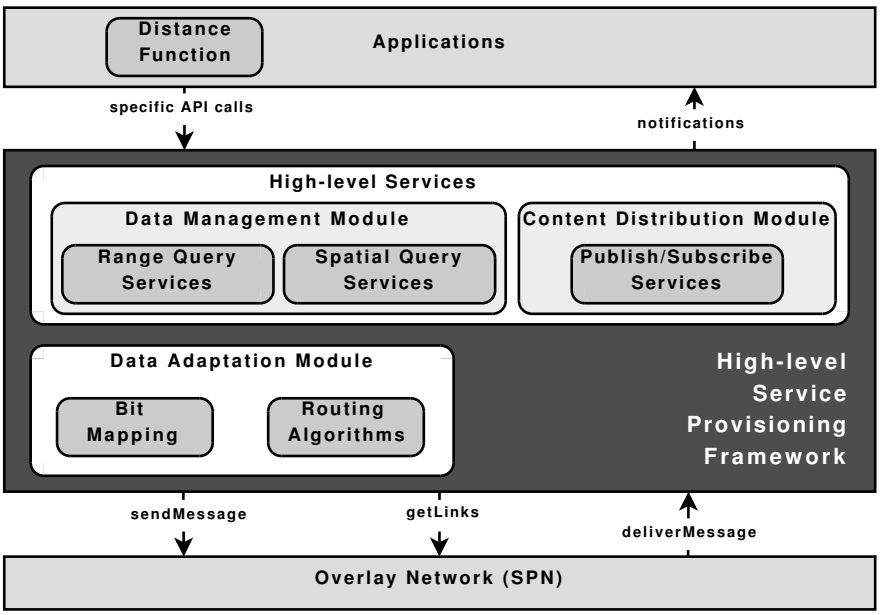


Figure 3.2: Structure of our framework provisioning high-level services. It also details all modules addressed in this work, as well as the information flow.

layer. However, to clarify the life cycle of the whole architecture, as well as its components, we will use an application example. We take a distributed image database as a target application. This application characterize any image as a M-dimensional vector of numerical features. The service required is k-nearest neighbor (kNN) queries, to find the most close images to the queried one. The Euclidean function is used to calculate the proximity between any two data objects. Let us now introduce all layers in the following lines.

Application layer. One could also see these **end-user applications** as clients of our framework. Not only should our framework be portable, but also it should support several end-user applications. Since the application data domain is specific for any application, the management of the data objects (like the data storage) is addressed by applications.

As we have seen in the state-of-the-art analysis, the distance function permits the evaluation of the similarity and closeness of any two data objects. Given that applications have the full knowledge of their own data domain, they make their particular *distance function* available to any service that requires it for its normal operation.

For instance, the image database would store locally the images that the local node is responsible for. When the application starts a kNN query in a certain node, the

application also provides the distance function, as well as the access to the images stored locally. This data access makes possible to the service to proceed the operation to the local objects. Thus, all services included in our framework are generic, so that services operate with the specific data domain and utilize the application's distance function. Let us illustrate how the query is followed into our framework.

Framework layer. Our **framework** appears in the second layer. This framework embraces most of the work done in this thesis. As we have listed in the motivation at Section 1.1, our framework should provide support for *high-level services*, as well as for *complex data domains*. To do so, briefly speaking, we design a first **module** for **data adaptation**, which adapts the application data domain to the SPN keyspace. This module also provides a range-based algorithm for complex distributed operations, benefiting to and reducing the effort put on featured high-level services.

Notice that a trade-off is presented when designing our data adaptation module from a software engineering viewpoint. According to the designed data structure, the necessary algorithms to resolve complex operations will become more or less expensive accordingly. For instance, let us suppose that we devise an efficient data placement technique (i.e., efficient data structure). Let us suppose also that the related algorithms to perform complex operations turn in very expensive distributed computations, though, which makes the whole approach rendering ineffective. Therefore, the aim is to equilibrate the complexity of both data placements and distributed algorithms (i.e., reaching a good balance between data locality, and data and routing load balancing), so that the distributed data structure could become efficient both in storage and operation execution. We explain the adopted approach for the data placement policy, common into the framework, in the following Section 3.3.

As part of the same layer, we include any existing high-level service. We exemplify the way of provisioning high-level services by developing three services. The **data management module** contains two services, affording range queries and spatial queries. We elaborate on them in Chapter 4. An additional module targeting **content distribution** techniques makes available content-based publish/subscribe services. They are illustrated in Chapter 5.

Following the image database example, the kNN query would be invoked to the corresponding service module. This module would use the data adaptation module, in order to adapt the query object to the SPN keyspace. After that, the service module would employ the necessary distributed algorithms to complete the kNN query. In

other words, the algorithm would start communicating with neighbor nodes, in order to distribute the kNN query. Whenever possible, the algorithm would parallelize the query to reduce the completion time, taking advantage of the node computation capabilities.

When a node receives the given query, the message is delivered to the same service module, so that the search operation can be processed locally. Not only could this process perform a search on the data objects of the local image database, but also it could forward the query to other neighbor nodes if necessary.

SPN layer. In this thesis we assume a set of common properties as described in Section 2.1.1. Broadly speaking, a SPN candidate has a uni-dimensional keyspace $\mathcal{I} = [0..2^m)$, where m is the precision in number of bits. In addition, the SPN is able to route between any two nodes with a logarithmic cost in the worst case, while maintaining an amount of routing state logarithmic in the number of nodes. By choosing a SPN as a substrate, thus, the framework benefits from its operation **correctness** and **efficiency**. In particular, in any of the three services presented in this thesis, we accompany our proposed solution with an instance of an SPN, suitable for the purposes of the approach. Even though they are different, the goal is to illustrate the **portability** of our framework.

To conclude with our example, the service module builds the necessary messages to distribute the query. These messages are forwarded using the SPN as a communication and transportation infrastructure. Since the query was adapted to the SPN keyspace, there is no difficulty to use it while the query is being processed by our distributed framework. The kNN search concludes with the result gathering at the querying node, using the SPN to send back as many results as necessary.

Once we have outlined the design of our framework and all its components, we delve into the data adaptation module in the following section.

3.3 Data Adaptation Module

As we have sketched, the data adaptation module tackles both (i) the adaptation of the application data domain to the SPN keyspace and (ii) the provisioning of a range-based algorithm for distributed data management. We address the former problem by the design of a novel linearization mechanism that enables structured data management without the burden of a global information maintenance scheme. The consequence of applying this linearization mechanism is that the resulting system sets

up the basis for range-based operations (like range queries), as well as individual addressing (such as exact match queries, i.e., $value \leftarrow get(key)$), even for complex multi-dimensional data objects. We detail how this data adaptation (or mapping) is addressed in Section 3.3.1. This adaptation technique is complemented by a range-based algorithm, whose goal is to facilitate the resolution of range-based operations in a distributed setting. We give their details in Section 3.3.2. In addition, we realize an analysis of the performance of this module in Section 3.3.3

Before starting with the description of the components of this module, let us introduce two functions used along this text.

Definition 3.1 (Number of dimensions) *The function $|\Delta|$ calculates the number of dimensions of Δ.*

The actual value for Δ can be either a data domain or any of the data objects from the data domain. In addition, because we will need to perform bit-wise operations, we introduce the following function:

Definition 3.2 (Bit precision) *The function $\| \Delta \|$ tells the number of bits necessary to represent Δ.*

This function can be only applied to uni-dimensional data domains and to their single values. Let us now detail the two kinds of data objects that the module can manage. Firstly, a vector of features of an image, a geographical location or an event are classified as *objects*. The idea behind an *object* is that it defines a *point* into the application data domain $\mathcal{O}$.

Definition 3.3 (Objects) *An* object *is any uni-dimensional or multi-dimensional data object $O = \{o_1, o_2, o_3, \ldots, o_D\}$ in the application data domain $\mathcal{O}$ (i.e., $O \in \mathcal{O}$), where $D = |\mathcal{O}|$ is the number of dimensions of the data domain, and each o_i is the single value for the i-th dimension.*

The second kind of object includes range query definitions and range objects, for instance. We call them *range objects*. In the literature, they appear named also as rectangular objects. Their main characteristic is that for any specified dimension, they define a selection of a range of values from the application data domain. That is, range objects defines a (hyper-)rectangle in the application data domain $\mathcal{O}$.

Definition 3.4 (Range objects) *A* range object *is any range-based data object $RO = \{[min_1 .. max_1], [min_2 .. max_2], \ldots, [min_D .. max_D] \}$ in the application data domain $\mathcal{O}$ (i.e., $RO \in \mathcal{O}$), where $D = |\mathcal{O}|$ is the number of dimensions of the application data domain $\mathcal{O}$,*

and each min_i and max_i determines the lower and higher bound, respectively, of the range selected on the dimension i.

In the following sections we detail our adaptation technique, the algorithm to perform range-based distributed procedures, as well as we illustrate on the performance of our adaptation technique.

3.3.1 Bit Mapping: Adaptation Function for Data Domains

As a key component of the framework, we provide our adaptation function called **Bit Mapping** (**BM**). One can see BM not only as the function $\mathcal{F}_{\mathcal{O}}$ firstly described at Section 2.1.1 "Common Properties of Structured Peer-to-Peer Networks", but also as $\mathcal{F}_{\mathcal{CO}}$ introduced in the publish/subscribe analysis in the Definition 2.12 "Complex object adaptation". Its goal is to transform objects from a complex (potentially) multi-dimensional data domain $\mathcal{O}$ to the SPN keyspace uni-dimensional $\mathcal{I}$. Broadly speaking, the adaptation function constitutes a two-phase process, where the first one transforms objects from the application data domain $\mathcal{O}$ to objects from an intermediate representation $\mathcal{O}'$, with the same number of dimensions than $\mathcal{O}$ (i.e., $|\mathcal{O}| = |\mathcal{O}'|$). The second stage ends the process by performing a dimensional reduction, where the intermediate object is transformed to a uni-dimensional value.

In addition, when designing BM, we leverage the rendezvous model from the underlying SPN routing. The goal of BM is to transform single and range-based data objects into one or more keys, whose responsible nodes will be defined as their rendezvous nodes. We elaborate on the adaptation approaches for both kinds of objects in the following lines. As we will see, BM enables to work naturally with *multi-dimensional data domains*.

3.3.1.1 Object adaptation

BM maps an object into only one key. Formally, let $O = \{o_1, o_2, \ldots, o_D\}$ be a D-dimensional object, where each $o_i \in \mathbb{Z}$ is represented by r bits, $i = 1, 2, \ldots, D$. Let $B = \{b_1, b_2, \ldots, b_D\}$ be a D-dimensional natural value, where each b_i defines the number of mapping bits per dimension, where each $b_i \in \mathbb{N}$, $i = 1, 2, .., D$. Let $K = \sum_{i=1}^{D} b_i$ be the sum up of the number of mapping bits. Let $m = \| \mathcal{I} \|$ be the precision of SPN keyspace $\mathcal{I}$ in bits, i.e., the SPN keyspace is of the form $[0..2^m)$ (see Assumption 2.1). The following inequality must be true for any B: $K \leq m$. This inequality ensures that the produced key falls into the SPN keyspace $\mathcal{I}$.

Our technique maps a D-dimensional object $O \in \mathcal{O}$ into a single key $k \in \mathcal{I}$ by means of a *monotone function* $P_{r,l}$. $P_{r,l}$ converts an incoming number of r bits into another one of l bits (i.e., $\| x \|_r \rightarrow P_{r,l} \rightarrow \| x' \|_l$). Recall that a *monotone* function is defined as follows: $f : S \rightarrow T$ is monotone, where each set S and T carries a partial order ($\leq$), if whenever $x \leq y$ then $f(x) \leq f(y)$. For instance, $P_{m,m-1}(y) = y \div 2$ is a monotone function, having S and T as $\mathbb{N}$, with a precision of m and $m-1$ bits, respectively. These kinds of functions are also so-called *order-preserving hash functions* (OPHF).

Intuitively, $P_{r,l}$ maps every o_i to a natural value of b_i bits, $i = 1, 2, .., D$, setting l by b_i correspondingly, and then we place it in the corresponding position into the produced key. For the sake of clarity, let us present the adaptation process split into the two stages, to afterwards present the *BM* formulation. Formally, let $\mathcal{O}' = \{W_1, W_2, \ldots, W_D\}$ be a data space where every W_i corresponds to a natural data domain $\mathbb{N}^+$ with precision of b_i bits, $i = 1, 2, \ldots, D$ (i.e., the domain W_i takes the form $[0..2^{b_i})$). Let BM_1 be the first stage in the adaptation process, which adapts the application data domain to an intermediate one:

$$BM_1(O) = \{P_{r,b_1}(o_1), P_{r,b_2}(o_2), \ldots, P_{r,b_D}(o_D)\} \tag{3.1}$$

Clearly, BM_1 transforms an object $O \in \mathcal{O}$ to an object $O' \in \mathcal{O}'$, where $O' = \{o'_1, o'_2, \ldots, o'_D\}$, and $o'_i = P_{r,b_i}(o_i)$, $\forall i = 1, 2, \ldots, D$. The next step is to proceed with the *dimensional reduction* from $D = |\mathcal{O}|$ dimensions to a single one (recall that $1 = |\mathcal{I}|$). Let '$\ll$' be the bitwise left-shifting function. Let $c_i = \sum_{j=1+i}^{D} b_j$, $\forall i = 1, 2, \ldots, D-1$, be the shifting factor. Let BM_2 be the second stage in the adaptation process:

$$BM_2(O') = (\sum_{i=1}^{D-1} o'_i \ll c_i) + o'_D \tag{3.2}$$

The BM_2's result is a numerical value of $K \leq m$ bits, which fulfills the dimensional requirements (both in number of dimensions and bit precision) of the target SPN keyspace $\mathcal{I}$ (i.e., if $O'' = BM_2(O')$, $O'' \in \mathcal{I}$). Let us now join both processes BM_1 and BM_2 into a single formulation, which concentrates the whole adaptation process in a unique function. To do so, let *BM* be our the mapping function:

$$BM(O) = (\sum_{i=1}^{D-1} (P_{r,b_i}(o_i) \ll c_i)) + P_{r,b_D}(o_D) \tag{3.3}$$

Thus, we can employ BM to map any multi-dimensional object O into another single one-dimensional key of $K \leq m$ bits. A direct consequence of this definition is that BM is a *deterministic function* (i.e., given the same $P_{r,l}$, B and object O, BM produces

always the same key) and that the system that employs it *can perform exact match queries for multi-dimensional values*.

Note also that the above definition carries some assumptions: (i) $O \in \mathbb{Z}^D$ and (ii) strict order of dimension mapping. One can overcome the former assumption easily, transforming values from other domains (e.g., floating point numbers or strings) into an equivalent integer domain. One can remove the later assumption by defining another D-dimensional set of numbers $ORD = \{ind_1, ind_2, .., ind_D\}$ which defines an explicit dimension mapping ordering. Therefore, ind_1 will be the first dimension to be mapped, ind_2 the second one, and so forth. We do not rewrite the definition of BM function because the addition of this explicit ordering will make its definition less readable, but always remaining the same operation.

3.3.1.2 Range object adaptation.

In a similar way than in the object mapping, BM maps range objects into a set of keys. Let $RO = \{[min_1 \,..\, max_1], [min_2 \,..\, max_2], \ldots, [min_D \,..\, max_D]\}$ be a D-dimensional range object, defining the *conjunctive* ranges of interest. Intuitively, instead of resulting a single mapping value per dimension, this technique will probably produce a set of different mapped values, representing all them the *mapped* range object. Clearly, we need the Cartesian product of all the sets of mapped values per dimension in order to cover the whole set of interests. Formally, we perform the first step of the adaptation process in the same way than before in Equation 3.1.

$$BM_3(RO) = \{[P_{r,b_1}(min_1)..P_{r,b_1}(max_1)], \ldots, [P_{r,b_D}(min_D)..P_{r,b_D}(max_D)]\} \tag{3.4}$$

The result of $BM_3(RO)$ is an object $RO' \in \mathcal{O}'$ of the form $RO' = \{[min'_1 \,..\, max'_1], \ldots, [min'_D \,..\, max'_D]\}$, where $min'_i = P_{r,b_i}(min_i)$ (resp. $max'_i = P_{r,b_i}(max_i)$) $\forall i = 1, 2, \ldots, D$. Let $RO_i = [min_i..max_i]$ and $RO'_i = [min'_i..max'_i]$ be the range interest of the i-th dimension on the range object RO or mapped one RO', respectively. Let $|RO_i|$ and $|RO'_i|$ be the number of different values in the range and mapped range, respectively. Let '$\bigotimes$' be the Cartesian product. To be congruent with the before object adaptation, RO' has to be passed through the Cartesian product of all its sets of values as follows:

$$\begin{aligned} BM_4(RO') &= RO'_1 \bigotimes RO'_2 \bigotimes \ldots \bigotimes RO'_D \\ &= \left\{ \left\{ min'_1, min'_2, \ldots, min'_D \right\}_1, \ldots, \left\{ max'_1, max'_2, \ldots, max'_D \right\}_n \right\} \end{aligned} \tag{3.5}$$

The result of $BM_4(RO')$ is RO'', a set of n single objects $O' \in \mathcal{O}'$. The last step in the adaptation process is to realize the *dimensional reduction* to all n objects O' from RO''. To do so, we employ the above function BM_2 from Equation 3.2. By performing this dimensional reduction operation, we obtain a keyset KS of n keys $k \in \mathcal{I}$ as follows:

$$BM_5(RO'') = \{BM_2(O'_1), BM_2(O'_2), \ldots, BM_2(O'_n)\} \tag{3.6}$$

Thus, KS is the set of keys built *deterministically* and corresponds to the original range object. Let us join all three steps into a single function, so that the BM function for range objects is as follows:

$$\begin{aligned} BM(RO) = \{k = BM_2(O') \mid \quad & k \in \mathcal{I} \quad \wedge \\ & O' \in RO'' \quad \wedge \\ & RO'' = BM_4(RO') \quad \wedge \\ & RO' = BM_3(RO)\} \end{aligned} \tag{3.7}$$

From the above definition, we can also quantify the number of covered keys as $|KS| = \prod_{i=1}^{D} |RO'_i|$. Since the responsible nodes of the keys in KS become the rendezvous nodes for the given range object RO, the $|KS|$ factor will greatly determine the cost of the range-based operation. We elaborate on the factors that influence in the costs of our adaptation technique at Section 3.3.3.

Let the *selectivity ratio* be the ratio that the *user selected* from the whole data domain (i.e., $^{(max_i - min_i)}/_{(domain\_max_i - domain\_min_i)}$ for the i-th dimension), where 0.0 (resp. 1.0) means that the 0% (resp. 100%) of the data domain has been selected.

Fig. 3.3 depicts a little example which explains intuitively how the keyset is obtained. The example consists on a 3-dimensional domain and a keyspace $[0..2^3)$, $m = 3$. The first step is the specification of a user *range object* $RO = \{[145..300], [-100 .. -50], [-200..400]\}$. The range object at the same time defines the *selectivity ratios* for each dimension, in this case 0.5, 0.5 and 1.0, respectively. Given the *data domains* and the 1-bit mapping per dimension, the next step is to proceed with the mapping process. We have designed for this example $P_{r,b_i}(y) = 2^{b_i} \times {}^{(y - min_i)}/_{(domain\_max_i - domain\_min_i)}$, which in fact is a monotone function. This function is applied as described above (e.g., $P_{r,1}(-100) = 2^1 \times {}^{(-100-(-100))}/_{(0-(-100))} = 0$), so that the range object is mapped into the set of values $RO' = \{\{1\}, \{0\}, \{0,1\}\}$, respectively.

The next step is to make the Cartesian product of these sets of values. This intermediate step constructs the set $RO'' = \{\{1,0,0\}, \{1,0,1\}\}$. Afterwards, we perform

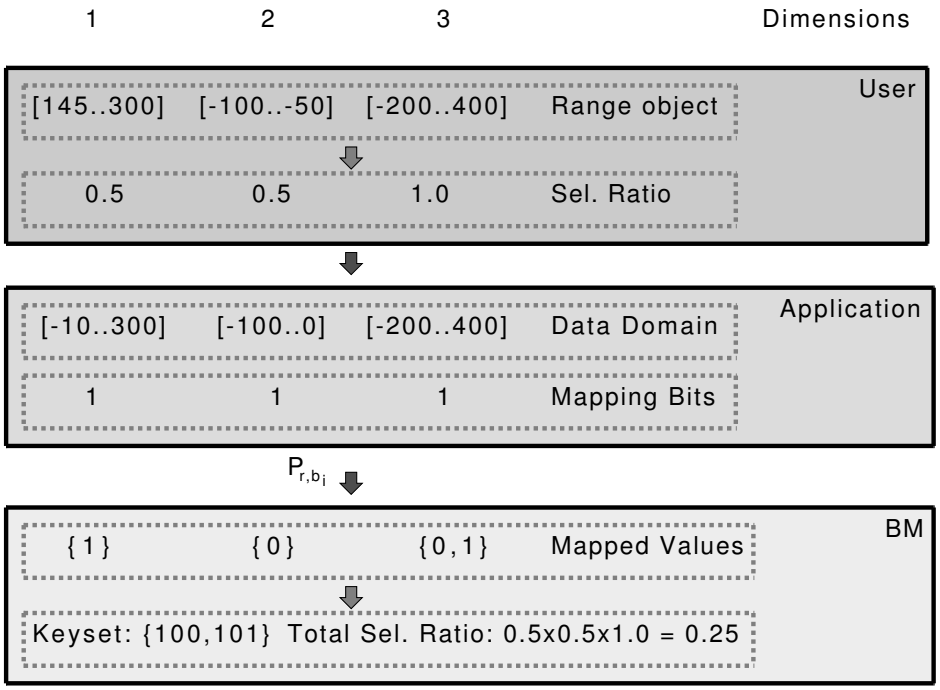


Figure 3.3: Example of adaptation of a range object.

the dimensional reduction, thus producing the keyset $KS = \{100, 101\}$, which concludes the adaptation process. Broadly speaking, in this example any key is the bit string produced by the concatenation of the mapped value from the i-th dimension into the i-th position. In addition, this keyset reflects the *total selectivity ratio* tied to the range object as the product of the given selectivity ratios (0.25), so that the final keyset KS consists of only 2 elements ($0.25 \times 2^3 = 2$).

Notice that our BM function have a particular behavior when constructing the keyset KS. Our adaptation function for range objects produces discontinuous disjoint segments of values. Let $h \leq D$ be the number of dimensions of RO where its mapped range object RO' has multiple mapped values. Let m be the number of mapped values of the last D-th dimension in RO'.

Remark 3.1 (BM's discontinuity on the keyset) *The BM function, when applied to range objects, constructs a keyset* $KS = \{k_{1,1}, k_{1,2}, \ldots, k_{1,m}, k_{2,1}, k_{2,2}, \ldots, k_{2,m}, k_{h,1}, k_{h,2}, \ldots, k_{h,m}\}$ *such that it can be represented as a set of different contiguous segments, i.e.,* $KS = \{[k_{1,1} \,..\, k_{1,m}], [k_{2,1} \,..\, k_{2,m}], \ldots, [k_{h,1} \,..\, k_{h,m}]\}$.

This property is very interesting since it simplifies the generation of the keyset. The reason behind that is because we do not need to calculate all different keys, but only the *minimum and maximum bounds* of the included segments of keys, which can be easily obtained from the result of the first step BM_3 (see Equation 3.4). This will greatly

alleviate the pre-process load of our BM function when deployed in our framework. This discontinuity is also useful to reduce the load when range objects have high selectivity ratios. That is, as we stated in Equation 3.7, the number of keys $|KS|$ is greater when the selectivity ratio of the range object grows. For instance, from the example appeared in Fig. 3.3, if the total selectivity ratio had been 1.0, the number of keys produces $|KS|$ would have been 8, the whole keyspace. Its effect is higher when the bit precision $\| \mathcal{I} \|$ of the SPN keyspace $\mathcal{I}$ is greater.

Remark 3.2 (Proportionally on the keyset's size) *The adaptation of a range object RO when using the BM function produces a number of keys proportional to RO's total selectivity ratio.*

Once we have explained how the data adaption is performed, we introduce in the following section the range-based algorithm that will help in the resolution of distributed operations on complex data domains.

3.3.2 Range-based Routing Algorithm

In this section we address how *range objects* are processed in the distributed system. To do so, we illustrate the procedure providing an suitable algorithm to process *range objects*. Note that since, for any given *object* $O \in \mathcal{O}$, $BM(O)$ produces only a single key $k \in \mathcal{I}$, the operations related to individual objects are strictly related to exact match operations (i.e., *put/get*). Complementarily, we define the Algorithm 3.1 that addresses the distributed process of range objects in our framework. The idea behind this algorithm is that local process of range objects at responsible nodes should come up with some local action, reacting to the incoming range objects (potentially sending back some message), as well as distributing them to other responsible nodes. To do so, Algorithm 3.1 leverages the rendezvous model broadly supported by SPNs. In other words, the goal of this algorithm is to support high-level services and to simplify their development by parameterizing the distributed process of range objects.

There are two important issues that we should consider while specifying this algorithm: (i) the underlying SPN properties and (ii) the effort on pre-processing the range object. We have already stated in Section 2.1.1 the characterization of the potential SPNs where our framework can be deployed in. Conversely, the second issue is very important while dealing with range objects, especially when range objects define high selectivity ratios. As we have outlined at Remarks 3.1 and 3.2, the adaptation of a range object produces a keyset with discontinuous disjoint segments of keys, as well

as it creates a number of different keys proportional to the range object's selectivity ratio, respectively. In order to provide a cost-efficient process of range objects, we do not construct the whole set of keys (i.e., the result of Equation 3.6 BM_5), but we only calculate the mapped values for each dimension (i.e., the result of Equation 3.4 BM_3), and proceed with the last steps Equation 3.5 BM_4 and Equation 3.6 BM_5 just on demand. Let us now introduce the Algorithm 3.1 that details the distributed management of range objects.

Algorithm 3.1 *range_object_management*
/* Distributed management of range objects. It allows to set up a particular action when the algorithm visits a rendezvous node. */
Input: $node \leftarrow$ Reference to the local SPN node
Input: $KS \leftarrow$ Selected keyset (i.e., $KS = BM(RO)$)
Input: $RO \leftarrow$ Range object to process
Input: $origin \leftarrow$ Reference to the node originating this process

```
1:  localKS ← segment(node, node) ⋂ KS
2:  if localKS ≠ ∅ then /* node is a rendezvous node */
3:     do the local action /* probably sending back to origin some message */
4:  end if
5:  remainKS ← KS \ localKS
6:  neighbors ← getLinks(node)
7:  while remainKS ≠ ∅ do
8:     rnode ← extract a node from neighbors
9:     rnodeKS ← segment(node, rnode) ⋂ remainKS
10:    if rnodeKS ≠ ∅ then
11:       range_object_management(rnode, rnodeKS, subs, node)
12:       remainKS ← remainKS \ rnodeKS
13:    end if
14: end while
```

Broadly speaking, this algorithm provides a way of multicasting a given information to several nodes (similar in essence to the work of S. El-Ansary *et al.* [50]). Lines 1-5 correspond to the local processing of the range object. Note that the *local action* (line 3) is not specified, but it is parameterized so that the high-level service could settle the corresponding operation. Line 5 establishes the remaining keyset *remainKS* to which *distribute* the range object. To do so, *node*'s neighbors (lines 6-14) are selected to forward the range object so that, from *node*'s viewpoint (lines 9-10), they can forward the range object or successfully process it. Note that this algorithm can visit nodes who are not responsible nodes (i.e., rendezvous nodes) of the keyset. If so, $remainKS = KS$

succeed (line 5).

The function $segment(p, n)$ determines the segment $[n\_min \,..\, n\_max] \in \mathcal{I}$ which node n is responsible for, using only the local information at node p where the algorithm is being executed. See Remark 2.2 for further details. The calculation of the responsibility's segment of the nodes (lines 1,9) helps on splitting the keyset KS into several disjoint segments, which ensures that the range object is going to be entirely processed.

Remark 3.3 (Multicasting service) *Algorithm 3.1 range_object_management can serve to distributed range objects, as well as any other objects, to a set of destinations.*

This algorithm could be invoked the first time as we illustrate in the following:

$$range\_object\_management(origin, BM(RO), CO, origin)$$

which would start the whole process, where *origin* is the node starting the operation. Note that CO could be any complex object that should be distributed, which usually would contain the referred RO. In addition, since $BM(RO)$ is actually a keyset, services can use this algorithm if necessary to send an arbitrary object CO to a set of destinations $\mathcal{T}$ (i.e., $\{k_1, k_2, \ldots, k_z\} = \mathcal{T} \subseteq \mathcal{I}$, where $z = |\mathcal{T}|$) as undermentioned:

$$range\_object\_management(origin, \mathcal{T}, CO, origin)$$

The most remarkable properties of this algorithm is its **low pre-process load**, **completeness**, **SPN-independent resolution**, **genericity** and the **one-way resolution**. In the former, as explained before, keyset operations (lines 1,5,9, 12) turn into *low cost* segment operations. By *completeness* we refer that the range object is completely *covered* by this algorithm just on the first node. The goal is to leverage the underlying SPN routing infrastructure by parallelizing the operation from the very beginning. As a key decision of our framework, we aim at providing a SPN-generic solution. To do so, we ask a minimum set of the node's local state information (such as $getLinks()$ or $segment()$ functions). In addition, the *path convergence* property of some of the SPNs produce no negative effect in our approach (e.g., overloading a path with copies of the same message), since a node can contact to any *neighbor* directly. Conversely, our framework also aims at supporting several services. This algorithm provides a *hot-spot* by defining the *local action* to perform on rendezvous nodes, so that this algorithm turns *generic*. The last property we consider from this algorithm is that nodes are not re-visited. That is, the algorithm ensures naturally that the range object is *covered* in

a *one-way* fashion. The reason behind that appears in the continuous splitting of the range object into *sub*-range objects (lines `5, 12`), so that nodes are not re-visited.

In the following section we realize a theoretical analysis of the cost of this algorithm, as well as the BM function. Experimental results by simulation are presented within the corresponding services' evaluation sections, which resemble to the following analytical results.

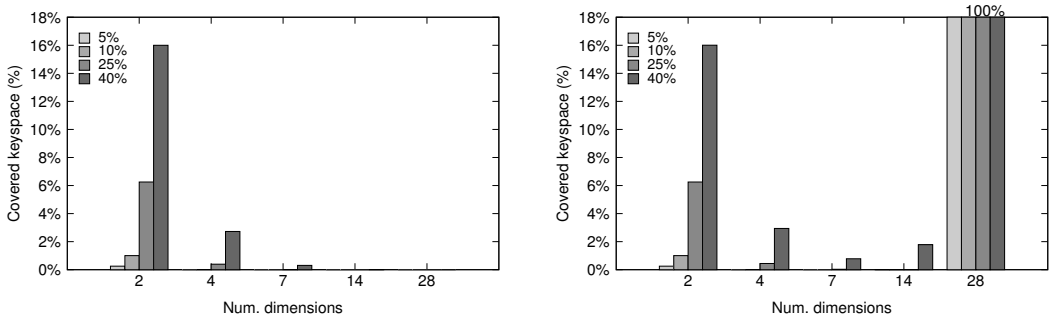
3.3.3 Data Adaptation Module: Evaluation

This is a theoretical evaluation of our data adaptation module, that addresses the analysis of two critical properties of the data adaptation module, namely, the *high-dimensional context property* and the *expected load from range objects*. Roughly speaking, the former establishes the basis that, for high-dimensional data spaces, our framework shows a better performance. The latter specifies the number of keys a range object is mapped into and, consequently, the amount of nodes which our algorithm must visit. We consider the nodes' load since, potentially, rendezvous nodes should perform some action on message reception, such as storing some information (e.g., the range object itself) or realizing some complementary operation (e.g., starting a new communication process).

3.3.3.1 High-dimensional context property

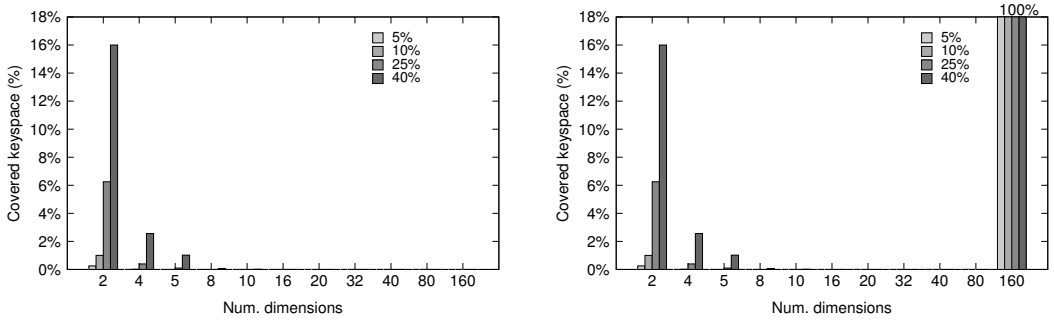
From Section 3.3.1.2, we see that BM maps a range object into $|KS|$ number of keys. Let s_i and b_i be the selectivity ratio and the number of mapping bits for the i-th dimension, respectively. Let $\sum_{i=1}^{D} b_i = K \leq m = \| \mathcal{I} \|$ succeed. Let $SR = \prod_{i=1}^{D} s_i$ be the *total selectivity ratio* of the range object. Thus, we can express the value $|KS|$ as a function of the total selectivity ratio SR, the number of range object's dimensions D and the keyspace size m. Specifically, $|KS| = \prod_{i=1}^{D} 2^{b_i} s_i = (\prod_{i=1}^{D} 2^{b_i})(\prod_{i=1}^{D} s_i) \leq 2^m SR$. Hence, BM maps a range object into $O(2^m SR)$ distinct keys. It is easy to see that whenever the total selectivity ratio or the number of mapping bits are high, $|KS|$ can be considerably big. In order to overcome this problem, the data adaptation module can attempt to modify any of the three involved variables: (i) reducing the number of mapping bits, (ii) reducing the selectivity ratio, or (iii) employing a high D-dimensional context.

The effect of **reducing the number of mapping bits** (i.e., $K' < K$) is to concentrate the whole range object coverage into a reduced segment of the keyspace $\mathcal{I}$ and, thus, unbalancing the load between nodes within the network. Instead, it is desirable that the number of mapping bits K be *as close as possible* to m in order to *balance the load*



(a) Lower bound **(b)** Higher bound

Figure 3.4: Lower and higher bounds of keyspace coverage by range selection mapping. $\| \mathcal{I} \| = 28$. Number of mapping bits per dimension *m/num. dimensions*.



(a) Lower bound **(b)** Higher bound

Figure 3.5: Lower and higher bounds of keyspace coverage by range selection mapping. $\| \mathcal{I} \| = 160$. Number of mapping bits per dimension *m/num. dimensions*.

through all nodes. The **selectivity ratio** is user dependant and, therefore, our framework cannot modify this variable. Nevertheless, low total selectivity ratios produce small sets of keys, regardless of the other factors. Instead, high total selectivity ratios produce mappings of (almost) the entire keyspace. The **number of dimensions** depends on the Application context and our framework cannot alter this factor. Nonetheless, suppose that we set every s_i to some selectivity ratio $\gamma < 1.0$, and that we increment the number of dimensions D to $D' > D$, the inequality $\gamma^D = SR_D > SR_{D'} = \gamma^{D'}$ comes true. Beforehand, thus, it is desirable a high-dimensional context to maintain the factor SR little. We call this property the *high-dimensional context property*. Figs. 3.4 and 3.5 depict this effect, where the keyspace covering is reduced while incrementing the number of dimensions. The explanation of these graphs are deeply provided in the following section.

3.3.3.2 Range object load

Let us analyse the effects of variating any of the aforementioned factors. If it is fixed that $\sum_{i=1}^{D} b_i = m = \| \mathcal{I} \|$, the number of dimensions can be modified and, consequently, the number of mapping bits per dimension, as we can see from Figs. 3.4 and 3.5. These graphs depict the *lower* and *higher bound* percentages of the keyspace covering for 5%, 10%, 25% and 40% selectivity ratios, having $m = 28$ and $m = 160$ bits, respectively. Note that the results of this coverage analysis also mean the expected amount of nodes that will potentially become rendezvous nodes along the processing of range objects. Additionally, note that while other systems detail their performance with very low selectivity ratios (e.g., between 0.1% and 3% in [82]), we perform a *deep analysis with wide ranges of selection*.

The lower (resp. higher) bound of $|KS|$ occurs when for any i-th dimension, s_i produces exactly $L_i = \lceil 2^{b_i} s_i \rceil$ (resp. $H_i = \lceil 2^{b_i} s_i + 1 \rceil$) distinct keys. Figs. 3.4a and 3.5a show the lower bounds for $m = 28$ and $m = 160$, respectively. Even though the covering ratio is at worst of up to 16% for the given selectivity ratios, the mappings of range objects when D grows produce smaller covering ratios. Recall that the smaller $|KS|$, the smaller amount of rendezvous nodes is required. For the sake of clarity, let us show up an example. Let $[0..3000]$ be the *price* data domain, where we define two range objects, $S_1 = [0..374]$ and $S_2 = [100..474]$, that have a 12.5% of selectivity ratio. Let $b_i = 3$ be the number of bits per dimension mapping. Consequently, $KS_1 = BM([0..374]) = [0..0]$, but $KS_2 = BM([100..474]) = [0..1]$, which are a clear example of lower and higher bound, respectively. In such cases, BM maps the range object into the following number of keys: $|KS| = \prod_{i=1}^{D} min(2^{b_i}, (2^{b_i} s_i) + 1)$. The min function limits the number of keys to maximum 2^{b_i}. W.l.o.g., let $(2^{b_i} s_i) + 1 \leq 2^{b_i}$ succeed $\forall i = 1, 2, \ldots, D$. The lower and higher bounds can then be formulated as follows:

$$\lfloor |KS| \rfloor = \prod_{i=1}^{D} 2^{b_i} s_i \leq 2^m \tag{3.8}$$

$$\lceil |KS| \rceil = \prod_{i=1}^{D} (2^{b_i} s_i + 1) \leq 2^m \tag{3.9}$$

In summary, our data adaptation module maps range objects into one or more keys, where its selectivity ratio has a great effect. In particular, we have that the following inequality $\lfloor |KS| \rfloor \leq \lceil |KS| \rceil \leq 2^m$ succeeds. In other words, BM maps range objects into $\Omega(\prod_{i=1}^{D} 2^{b_i} s_i)$ or $O(\prod_{i=1}^{D} (2^{b_i} s_i + 1))$ *distinct keys* and they express the *lower and higher bounds* (i.e., the best and worst case), respectively. Figs. 3.4b and 3.5b depict this higher bound.

Besides, it is easy to see from the graphs that our mapping technique is independent of m, the keyspace precision. Nonetheless, the greater m, the higher dimensional contexts our framework supports. From another viewpoint, as announced by the *high-dimensional context property*, the covered keyspace by range objects of $b_i \geq 2$ is very reduced as long as the number of dimensions increases. In addition, low dimensional contexts ($D \leq 5$) experience less than 16% of keyspace covering for any of the selectivity ratios. This fact produces that, actually, very few nodes are selected as rendezvous nodes, alleviating the overall system load.

Nevertheless, our framework would not scale generally in scenarios with 1-bit mappings and 2-bit mappings with very high total selectivity ratios. Note that every one-bit mapping value (i.e., 0 and 1 are the possible values) maps the 50% of the domain to each bit value, providing poor precision. For this reason, we do not include 1-bit mappings in the simulation scenarios of our service use cases.

3.4 Conclusions

In this chapter we have introduced our framework in Section 3.2, which arranges the basis for a *high-level service provisioning*, *regardless of the underlying SPN*. In addition, we construct the framework in such a way that *complex data domains* are supported. To harbor complex data domains though, we settle down the **BM function** (in Section 3.3.1), as well as an **algorithm** to support range-based operations (in Section 3.3.2), within the data adaptation module (in Section 3.3).

As we have seen, the BM function constitutes the key element into the data adaptation module. BM is a *deterministic* mapping function. Clearly, a certain object (resp. range object) is mapped to the same key (resp. set of keys), independently of the node who realizes the operation. For instance, given a certain network setting and that an object O is mapped into a key k whose owner node is p, any node n will send the object O to the rendezvous node p, responsible node for k. The same occurs with range objects. Our Algorithm 3.1 leverages the rendezvous model to meet the keys (e.g., from range object mappings) with their responsible nodes (i.e., rendezvous nodes), where some local operation is requested to perform. To implement this algorithm, we use a minimum subset of the common SPN state information, such as the set of neighbors.

Finally, we have evaluated our data adaptation module against a theoretical analysis of the critical properties that influence in the performance of our adaptation technique. These results establish the minimum bound for the results that will be depicted in the services to be described in the following chapters. The most important property

is that our framework *suits better for high dimensional data spaces*, as we have seen in Section 3.3.3.1. Even though the main two factors that limit the good performance on the data adaptation and range-based operations are the selectivity ratio and the keyspace precision. The former is user dependant and cannot be controlled. However, it is easy to see that for high selectivity ratios, the number of rendezvous nodes will grow, restricting the performance of our approach. The latter confines the number of adaptable dimensions to the keyspace, yet it is not under the framework's control. For instance, with a 160-bit node identifier (like in Chord [7]), our framework would tolerate up to 80 dimensions. That is, the adaptation technique necessitates a minimum of 2 bits per dimension.

With this chapter we have illustrated not only our framework, but also the data adaptation module. This module consists the BM function and the algorithm supporting range-based operations, which illustrate and validate the second and third contributions of this thesis. For further validation, we instance our framework and the data adaptation module in three different services in the following chapters, establishing the soundness of our design.

4

Multi-dimensional data management

This chapter describes how our adaptation technique can be used to provide data management services, constituting the core of our **data management module**. In particular, we detail two use cases out of most demanded services. The range query services are presented in Section 4.1. In addition, we worked toward the provisioning of geographical location services, illustrated in Section 4.2.

4.1 Supporting range queries

Similarity search is a hot research topic on peer-to-peer systems. In this section we present SQS, a **s**imilarity **q**uery **s**cheme for peer-to-peer databases. In this approach we employ our novel linearization mechanism seen in Section 3.3, that enables structured queries without the burden of a global information maintenance scheme. The system offers exact match and range searches of multi-dimensional data. To illustrate, SQS is instantiated in Cyclone, a hierarchical overlay that is able to build disjoint clusters in terms of network latency, and enables load balancing on searches by caching in a per cluster scheme. We conclude this use case demonstrating the good properties of SQS through representative simulation results.

4.1.1 Introduction

Earlier peer-to-peer proposals were unstructured overlays, like Gnutella [6]. Their communication scheme was based on flooding mechanisms, incurring on an important amount of overhead. Later, SPNs and the DHT paradigm emerged (e.g., CAN [11], Chord [7], Pastry [8]), providing exact match queries with good performance qualities. These systems perform insertion and lookup of data via consistent hash functions, thus guaranteeing a uniform data distribution over the network.

Nevertheless, real-life applications demand more complex searches, such as *range queries*. These kinds of searches enable the user retrieve related or *similar data objects to the query*. Nevertheless, SPNs do not deal with these kinds of searches. Instead, due

to the use of consistent hash functions, we would need to flood the entire network for retrieving all related content, which becomes unacceptable for large-scale networks.

All these (distributed) applications present the same elements in their architecture: a **multi-dimensional data domain** with data objects being stored and searched; a **distance function** which defines the similarity between two data objects; and the **algorithms** for storing and searching the content. For instance, consider an image querying system where users publish images. These images are characterized by real-valued *D-dimensional feature vectors*. A query in this system consists of such a vector and the user expects the most similar images to it. Indeed, Information Retrieval (IR) applications work in the same manner, where each text document is characterized by a *D-dimensional vector* with the best descriptive terms. In both cases, when a search is performed, applications must evaluate the similarity of various data objects according to some *distance function*, e.g., Euclidean distance for image retrieval applications or Cosine distance for text retrieval ones, as well as each (distributed) application decides *how and where to store* the content.

In particular, our approach aims at *supporting efficiently similarity searches for a wide range of applications in large scale peer-to-peer systems*. To do so, we leverage our *data mapping scheme* and the related *set of algorithms for storing and searching the content*, which operates without the burden of a global information maintenance scheme (see Section 3.3). This work provides room for semantically different applications, to which we provide similarity query services. Indeed, these applications can be collocated within the same SPN and operate efficiently. It is easy to see that the users of our similarity query services are the specific applications, such as image or document retrieval applications.

Nevertheless, there are some technical challenges that we have to address prior to constructing the similarity query services:

- **Efficiency on content placement and search.** Data objects must be placed taking into account the overlay network structure to *get the benefits of its routing properties*, such as the network diameter.

- **Latency-aware operation.** Large-scale distributed systems should afford a low communication cost, for example, in terms of *latencies* and *number of hops*. In our opinion, an overlay network that fits better with the underlying Internet architecture suffers from less communication penalties.

- **Load balancing.** The state information in the system must be manageable (i.e., list of neighbors, routing table and data indexes). In addition, both storage and computation costs should be approximately the same for all nodes.

- **Consistency.** A system is *data search consistent* when content is always found whenever this is available in the system. A certain level of consistency may be guaranteed via caching and replication on any system, but then this becomes more loaded and, hence, less efficient.

- **Multi-domain support.** The system has to provide transparently both *multi-domain support* and *space-mapping* to applications. The former enables different contexts for applications (i.e., a database for each application). The latter addresses how the application domain is mapped onto nodes; in other words, it shows where a multi-dimensional object is placed in the network, in such a way that the overall efficiency is guaranteed.

Our SQS system addresses all the challenges described above. In particular, we view the following issues as the main contributions of this work:

- We provide with SQS a novel scheme for mapping the content to the nodes within the network, based on our Bit Mapping function (BM) (see Section 3.3.1). By this scheme we can offer a common search algorithm for answering similarity queries efficiently, but personalized to each application.

- We use Cyclone [42] as the overlay network. Cyclone is able to build a hierarchy of disjoint clusters of nearby nodes, for instance, in terms of latency. Besides, we leverage Cyclone's properties, so that we efficiently exploit its hierarchy for guaranteeing load balancing throughout the clusters.

- SQS provides multi-domain support. That is, we offer these similarity search services simultaneously to several applications, guaranteeing separately their consistency.

The key issue behind SQS resides on the BM mapping scheme that defines where to record data objects in a peer-to-peer database, and also defines a set of algorithms to perform efficiently similarity queries. Broadly speaking, this mapping scheme builds the key to be routed through the overlay network, having as its input the vector that describes any multi-dimensional data object. Searches are processed in a similar way so that related objects are found.

The presentation of this work is structured as follows. In the following section we make a succinct review on the related work of our approach. In Section 4.1.3 we introduce our range query service, with the description of the drawn SPN and the construction of our high-level service. We evaluate our approach in Section 4.1.4 whose results prove the soundness of our design. We give some concluding remarks about SQS in Section 4.1.5.

4.1.2 Related work

There exist several works in similarity search in the P2P field. We have stated an extended analysis in Chapter 2, where, in particular, we reviewed P2P systems providing range queries (Section 2.5). Nevertheless, for the sake of clarity, we make in the following lines a concise review of the related work in the field of this work.

Earlier works such as PHT [55] and SkipNet [26] only offer range queries for one-dimensional datasets. Other systems support multi-dimensional similarity searches, even though they address partially the challenges described above. Some examples are SkipIndex [28] and the work of Bin Liu *et al.* [57].

Some of existing systems operates with a multi-dimensional domain to identify nodes and can map multi-dimensional content onto nodes with a little effort [29, 97]. Because most SPNs use one-dimensional keyspaces to identify nodes, these solutions need some mechanism to map multi-dimensional content into the network. Some works, such as [26, 29, 97], use linearization functions like SFCs [33] instead of SPN's consistent hash functions. This kind of linearization function, also called order-preserving hash functions or locality-preserving hash functions, addresses getting similar content distributed near the same place into the network. By distributing the data this way, the goal is to visit only few nodes for any arbitrary search. But linearization functions and related querying algorithms are oblivious to the underlying SPN topology. Thus, semantically close nodes may be far in terms of communication cost. Our approach, however, is aware of the underlying SPN and benefits from Cyclone's overlay topology. By leveraging Cyclone, we are able to reduce communication costs. Above all, the most similar to this work is ZNet [32], so that we evaluate SQS against ZNet in Section 4.1.4.

ZNet is based on Skip Graphs and uses SFCs to map multi-dimensional data objects to nodes. ZNet and this work are similar in the following terms: (i) the mapping scheme builds a logical hierarchy, defining where to place the content; (ii) this scheme enables nodes to easily check whether they own content related to a similarity query, by comparing the value of their node identifier and the query; and (iii) consecutive

nodes in the node identifier domain have contiguous data domain fragments. In ZNet, nevertheless, data is mapped onto a flat overlay network and data load balancing is addressed by joining and rejoining nodes into heavily loaded network zones, which becomes an excessive management cost. This data management mechanism prevents from a real scalability and feasibility of the solution. Eventually, ZNet does not deal with network communication costs and can support just one application at a time.

4.1.3 SQS: the Similarity Query Scheme

SQS aims to be a large scale system for similarity searches. As we can see in Fig. 4.1, Cyclone appears in the bottom tier, which supports the SQS services in the upper tier. SQS similarity services are built from two main building blocks, to wit Storage and Search services. To provide them, SQS uses the Bit Mapping services to adapt and process both data objects and queries. SQS differs from other systems because it leverages the underlying overlay topology to boost the query performance. In the top tier, specific Applications use our similarity search scheme and benefit from our distributed, scalable solution.

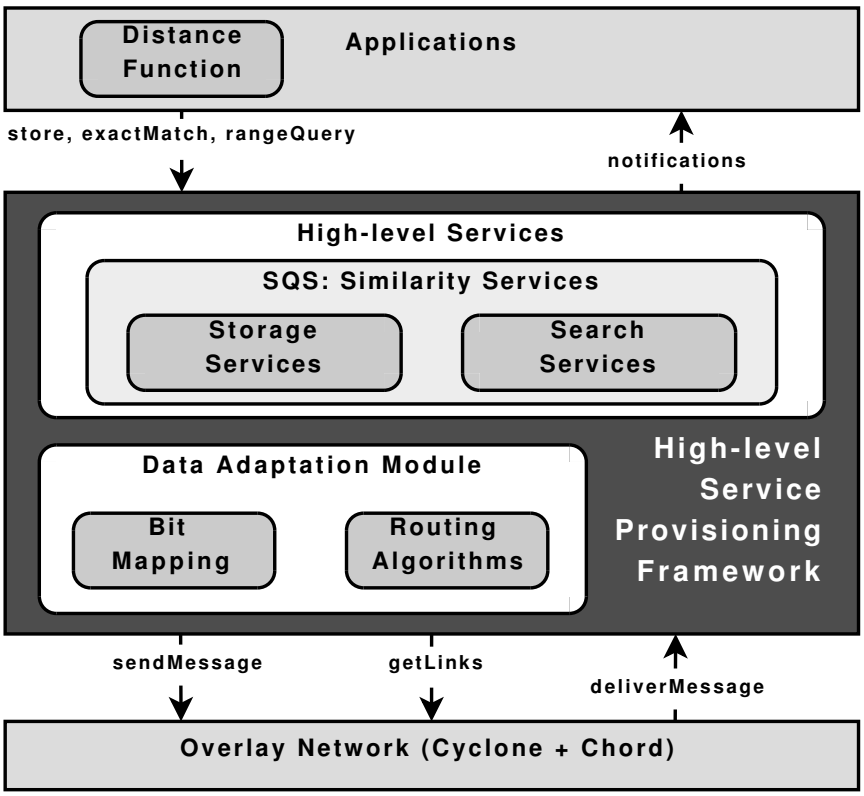


Figure 4.1: SQS Architecture

4.1.3.1 Electing the Routing Infrastructure: Cyclone

In this work we have selected Cyclone [42] as the SPN substrate where to deploy our range query services. Rather than a specific overlay network, Cyclone is a framework for arranging a set of structured disjoint overlays into a hierarchy, without imposing a specific overlay topology for the leaf clusters. In this case, as a proof of concept, we have selected Chord [7] as the *instantiated* overlay within Cyclone. The reader can find an example of a Cyclone architecture in Fig. 4.2.

An important feature of Cyclone is that it is able to organize *close* nodes into hierarchical clusters, where the *closeness* can be any arbitrary node's characteristic (see Remark 2.7). In our approach, we leverage Cyclone's properties to construct a SPN efficiently organized to minimize the network delay, exploiting the physical proximity of the underlying network. Further, it also provides some fundamental features like load balancing, replication, content caching and fault tolerance.

For the sake of clarity, we present some necessary definitions in order to facilitate the reading and comprehension of the rest of the work:

- Having a b-bit SPN keyspace, the cluster identifier *clusterId* is formed by the cl less significant bits (LSB) (i.e., a suffix of cl bits), $cl < b$ and $cl \geq l$, where l is the cluster level (illustrated in Fig. 4.2a).
- The node identifier *nodeId* for a node at l'th hierarchy level is formed by the most $b - cl$ significant bits (MSB) (i.e., a prefix of $b - cl$ bits). This *nodeId* is used within the intra-cluster overlay protocol (Chord in this case).
- A node p is tagged "$EP^{k}_{C_n}$", namely C_n's exit point for a key k, whenever p is responsible for k in the cluster C_n.

Indeed, the following properties are the most remarkable attributes of Cyclone:

- By organizing nodes in terms of network latency, Cyclone significantly outperforms a traditional flat SPN (illustrated in Fig. 4.2b).
- Cyclone enables to build a hierarchical system with manageable state information, taking only $O(\log|\mathcal{C}|)$ inter-cluster shortcuts, where $|\mathcal{C}|$ is the number of leaf clusters present in the hierarchy.
- The diameter for accessing between any two different clusters is yield to Δ hops, where Δ is the number of bits for the binary representation of $scId \times tcId$, being $scId$ and $tcId$ their $clusterIds$, and $\times$ the bitwise exclusive OR.

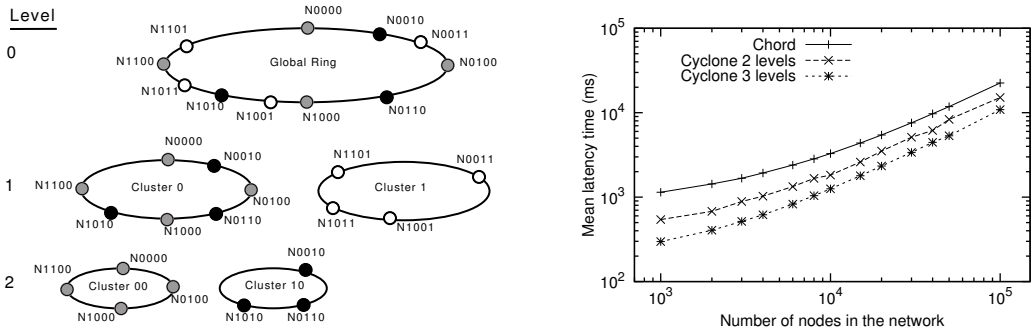


Figure 4.2: Cyclone architecture. **(a)** Example of a Cyclone setting, describing the location of nodes within the clusters. The instantiated SPN is Chord. **(b)** Communication cost in a flat Chord against the 2- and 3-level Cyclone with Chord as instantiated SPN. The simulation scenario is as follows. GT-ITM topology with 100-node highly connected backbone. Latency weights: 10ms for backbone edges, 100ms for backbone- stub edges and 5ms for stub-stub links. To construct the network with the desired size, we attach a suitable number of Cyclone nodes to each stub node assuming a latency of 1ms for these edges.

- Cyclone's lookup algorithm takes advantage of the network locality, firstly looking up in leaf clusters and then, if necessary, forwarding greedily the query to a foreign cluster, via the exit points, until the responsible node is reached.
- Exit point nodes are elected for data caching and replication, achieving an efficient search load balancing. Intuitively, the first lookup will retrieve the data object from the owner (cluster-foreign) node, but extra lookups will match in the local cluster's exit point nodes.

For further details about Cyclone, we refer the reader to the background, in Section 2.1.2.1, where we have described it, or to have a look at [42] for full details about this hierarchical peer-to-peer network.

4.1.3.2 SQS Services

We have designed storage and search algorithms to provide efficiently similarity search services for a wide range of applications. Particularly, SQS supports exact match and range searches for multi-dimensional data, based on the Algorithm 3.1 for range-based operations. At a glance, the entire process for an insertion or search operation is the following.

Insertion and search operations are started at Application tier. The Application establishes the data object and related data to be inserted or the range query to be

searched. Data objects must be *objects*, to wit point data objects. Nevertheless, *range objects* like (hyper)rectangles may be inserted, transforming them into high-dimensional *objects*.

Afterwards, SQS firstly maps either the data object or the range query into the SPN keyspace. We base the mapping of this multi-dimensional data on our *data adaptation module*, specifically in our Bit Mapping (BM) function (see Section 3.3). Basically, BM constructs a single key or a keyset from an object or a range query, respectively. Since data objects are mapped into a single key, the corresponding messages are trivially routed through the overlay routing infrastructure to the responsible node. The same is not true for range queries though.

We define a *parallelized breadth-first-like range search algorithm*. This algorithm is depicted in Algorithm 4.1, based on Algorithm 3.1, which also takes into account the hierarchical structure of Cyclone. To illustrate why, consider a range query RQ, which is disseminated into sub-queries through neighbors of the querying node. Whenever the sub-queries cannot be answered from within a single Cyclone's cluster, nodes route them through the Cyclone hierarchy. Conversely, whenever the search strictly relies in a single cluster, nodes of this cluster take advantage of the intra-group low communication cost. Remember that nodes are organized by network latency. That is, the deeper the cluster level, the closer the nodes. Therefore, the query resolution algorithm benefits from Cyclone and its latency-aware node organization to boost the process, reducing the communication latency. This effect was already visible in the comparison of Cyclone against a flat Chord in Fig. 4.2b.

As we can see from Algorithm 4.1, responsible nodes (i.e., rendezvous nodes) look for objects fulfilling the range query predicates into the set of locally stored data objects $\mathcal{O}'$ (line 3). The percolation rule for objects to be selected is specified by $\mathcal{M}_{\mathcal{Q}}$, which is defined as follows:

Definition 4.1 (Distance function for objects and range queries) *Let the application domain space $\mathcal{O}$ be D-dimensional. Let RQ be a range query defined in such a data space. Let o be a data object to check if it fulfills the RQ's predicates. The function $\mathcal{M}_{\mathcal{Q}}$ establishes the distance between a range query and an object as follows:*

$$\mathcal{M}_{\mathcal{Q}}(RQ, O) = \begin{cases} 0 & if \forall i = 1, 2, \ldots, D : RQ_i.lowerBound \leq o_i \leq RQ_i.higherBound \\ \neq 0 & otherwise. \end{cases}$$

Actually, $\mathcal{M}_{\mathcal{Q}}$ is the *distance function* provided by the Application to our services in order to calculate not only the containment of objects into a given range query, but also distances between objects. Notice that when two objects are placed in $\mathcal{M}_{\mathcal{Q}}$ (i.e.,

Algorithm 4.1 *range_query*
/* Distributed resolution of range queries. */
Input: $node \leftarrow$ Reference to the local SPN node
Input: $KS \leftarrow$ Selected keyset (i.e., $KS = BM(RQ)$)
Input: $RQ \leftarrow$ Range query to process
Input: $origin \leftarrow$ Reference to the node originating this query

```
1: localKS ← segment(node, node) ⋂ KS
2: if localKS ≠ ∅ then /* node is a rendezvous node */
3:    answer ← {o ∈ O' | M_Q(RQ, o) = 0}
4:    if answer ≠ ∅ then
5:       send(origin, answer, RQ, node) /* (partial) answer to the query */
6:    end if
7: end if
8: remainKS ← KS \ localKS
9: neighbors ← getLinks(node)
10: while remainKS ≠ ∅ do
11:    rnode ← extract a node from neighbors
12:    rnodeKS ← segment(node, rnode) ⋂ remainKS
13:    if rnodeKS ≠ ∅ then
14:       range_query(rnode, rnodeKS, subs, node)
15:       remainKS ← remainKS \ rnodeKS
16:    end if
17: end while
```

$\mathcal{M}_Q(o1, o2)$) the same rules applies (having $RQ_i.lowerBound$ and $RQ_i.higherBound$ been replaced by $o1_i$, so that the condition comes as $o1_i = o2_i$). When an object o is not selected by a range query RQ (resp. does not equal another object $o2$), the distance function determines numerically in an Application-specific fashion the *closeness* between them. Note that we only use $\mathcal{M}_Q$ as a *boolean* function, even though other services like top-K queries or k-nearest neighbors would rely on the differences to locally classify objects for the given query.

Storage and Search Services

Specific Applications have their multi-dimensional data domain and must provide to SQS the monotone mapping functions and the set of mapping bits B, which are necessary for our BM function to work properly. SQS then employs the BM function to adapt objects and queries into the SPN keyspace, so that SQS is able to provide storage and search services. The reader can observe the relation of the provided services in

Table 4.1.

Storage service. We provide to Applications the function:

$$store(appName, attributeName, dataObject[, relData])$$

This function enables to store content related to each *appName* Application individually. Further, we provide flexibility to applications letting them to store as many kinds of content as they need, easily specifying different *attributeName*s. In the end, this function applies the BM function to the multi-dimensional *dataObject*, resulting in a mapped key $k = BM(dataObject)$. A message is built with all this information and routed through Cyclone. The Application standing on the Cyclone node responsible for k will store *dataObject* and the related content *relData* if present. The idea behind *relData* is to allow the Application to associate any kind of data to a given data object for a later lookup. Recall that the total number of mapping bits is desirable to be (nearly) the same than the number of bits of the node identifier domain in order to balance the data storage load.

Search services. We provide both exact match and range searches within SQS. The exact match search:

$$lookup(appName, attributeName, dataObject)$$

operates like the *store* service, but retrieving the *dataObject* and the related information *relData* if available. The main focus of these services relies on the range query, via the function:

$$rangeQuery(appName, attributeName, predicates)$$

where predicates are expressed as $\{[min_1..max_1], \ldots, [min_D..max_D]\}$ for a D-dimensional data space. We employ the tree-based BM search algorithm described above and applied onto the hierarchical Cyclone. Each node that has data objects selected by the range search answers with all records of the form $\{dataObject[, relData]\}$. We adopt individual instead of cumulated results from nodes to build manageable messages in terms of their size, as well as to provide promptly search results.

4.1.4 Similarity Query Scheme: Evaluation

In this section we conduct the evaluation of SQS. The goal of this analysis is to show the feasibility of our distributed approach, as well as the suitability of our distributed algorithms for the range query resolution. To do so, we present two separated evaluations of our approach. The first one introduces the efficiency and scalability of SQS.

Table 4.1: Provided SQS storage and search services.

Storage services
store(appName, attributeName, dataObject [, relData])

Search services
lookup(appName, attributeName, dataObject)
rangeQuery(appName, attributeName, predicates)

Afterwards, we compare SQS against ZNet [32]. As we have seen in the related work (Section 4.1.2), ZNet is the most similar work to our solution. In both cases, we reckon the number of *routing nodes* as the main property for the scalability of the given approach, since it is strictly tied to the distributed solution design. Recall that routing nodes are such nodes that are visited as part of the query routing process, but that they do not contribute with results to the query. We refer to this overhead as **noise**.

We obtained the evaluation results by simulation means. The default simulation settings, otherwise noted, are as follows:

Table 4.2: List of parameters of the simulation settings.

Parameter	**Value [default]**
$\|\mathcal{I}\|$	24 bits
N	1K..100K nodes [8K]
Num. dimensions	2..20 [8]
Num. mapping bits	12..1 [3]
Dataset size	300K data objects
Num. queries	20K

We have set up the size of keyspace $\mathcal{I}$ to $|\mathcal{I}| = 24$ bits to get the simulation results in a suitable response time. Given the 24 bits of the keyspace $\mathcal{I}$, we set up the number of mapping bits as the maximum value *mb* that accomplishes the following inequality: $num.dimensions \times mb \leq |\mathcal{I}|$. For instance, in an 8 dimensional data space, the corresponding *mb* value is 3 (as seen in Table 4.2). We have conducted the simulation by using synthetic datasets from 2 to 20 dimensions with two different data distributions: a uniform and a skewed one, based on a normal distribution. We have addressed the simulation by realizing 20K queries, where 100 randomly selected nodes performed 200 range queries. Since the results were very similar for both data distributions, we only present the results from the uniform one.

Efficiency and suitability of SQS We address the SQS performance evaluation measuring the number of *routing nodes* (i.e., the *noise*). In addition, we also account the algorithm **efficiency ratio**, which exhibits the ratio of routing nodes related to the total number of visited nodes:

$$\text{efficiency ratio} = \frac{\text{num. rendezvous nodes}}{\text{num. rendezvous nodes} + \text{num. routing nodes}}$$

This evaluation is addressed via simulations with up to 100K nodes. As depicted in Fig. 4.3, the number of routing nodes grows logarithmically with the network size. Conversely, the algorithm efficiency ratio is kept almost constant in the little range 78%..88%. In other words, our range query algorithm only reports from 22% down to 12% of noise in terms of routing nodes. This occurs because both the mapping scheme and the query distribution scheme of the algorithm nicely fit the Cyclone architecture. Otherwise, the presented routing efficiency ratio would be worse. However, the efficiency does not grows linearly, but sublinearly, because our range-based routing algorithm requires a minimum set of nodes to reach rendezvous nodes. This is not only a proper issue of our range query algorithm, but also a tightly related one to the underlying SPN routing scheme (Cyclone in this case).

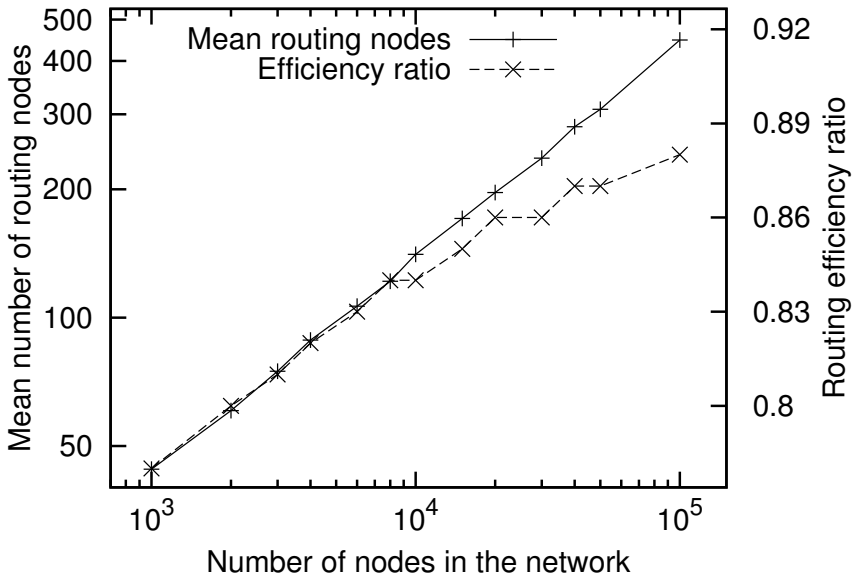


Figure 4.3: Range query analysis, comparing the number of routing nodes (i.e., noise) against the efficiency rate. Num. of dim.: 8. Num. of mapping bits per dim.: 3.

SQS vs. ZNet evaluation. To evaluate both systems we consider the four factors that are involved in range queries and that should be analysed: **data dimensionality**

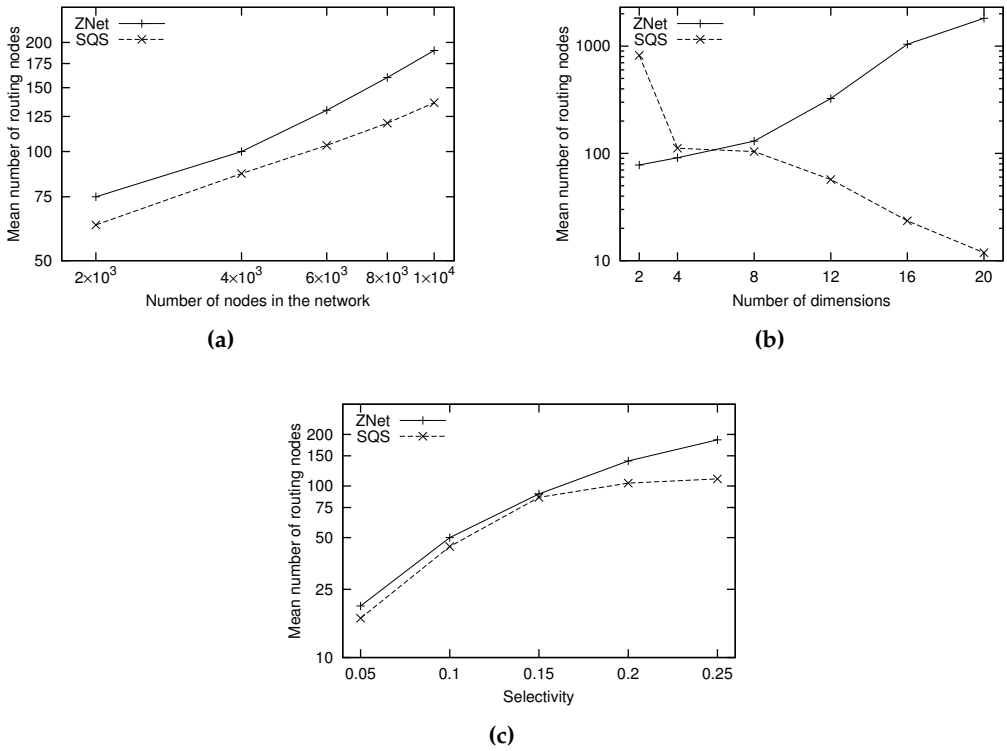


Figure 4.4: Range query evaluation between SQS and ZNet systems. Default simulation settings: 8-dimension data space; 8K-node network; 20% of selectivity ratio. The network size (**(a)**), the number of dimensions (**(b)**) and the selectivity ratio (**(c)**) are considered.

and **distribution**, **range selectivity** and **network size**. To do so, we evaluate our solution SQS against ZNet [32] in a common evaluation scenario with up to 10K-node networks. The results of this evaluation appear in Fig. 4.4 and present the number of routing nodes according to the simulation settings.

Since uniformly distributed datasets presented similar behavior to skewed ones, we only include the results from the first datasets in Fig. 4.4. Therefore, we only consider into analysis the *network size*, data *dimensionality* and *range selectivity* in Fig. 4.4.

We can observe in Fig. 4.4a that in both systems the number of routing nodes increases with network size. While in ZNet the noise raises almost linearly to the network size, SQS suffers from a logarithmic increase as depicted in Fig. 4.3. The reason behind that is because of our BM function and the range query algorithm. Since we assume a uniform distribution of nodes within the keyspace, and considering that the BM function maps a range query into discontinuous disjoint segments of keys, more rendezvous nodes lie in the same segment than in ZNet, so that the number of routing nodes remains almost unaffected in our approach.

In Fig. 4.4b we can observe that as long as the number of dimensions rapidly increases, so does the number of routing nodes in ZNet. Conversely, due to the *high dimensional effect*, the range queries deployed on SQS on higher data spaces are mapped to smaller keysets. This way, reducing the mapped keyset size, we also decrement the number of potential rendezvous nodes, so that the total cost of the resolution of range queries in SQS notoriously decreases. For the same reason, SQS does not suit for very low dimensional data spaces (see dimensionality 2 from the graph).

The last Fig. 4.4c analyses the effect of the selectivity ratio on the range query. SQS outperforms ZNet in all cases. In fact, given a network size and incremental range selectivity, SQS does not increase the number of routing nodes in the same order of magnitude than ZNet, due to the SQS tree-based search algorithm. To illustrate, notice that the effect of augmenting the selectivity ratio in SQS is the definition of wider segments of keys. In consequence, visiting neighbor nodes of a rendezvous node within a certain segment do not incur on additional routing nodes, as long as they are also rendezvous nodes.

Load balancing. In this analysis we have not addressed the issues related to the data distribution, such as the data distribution and the corresponding data load into nodes. In addition, we have deployed no caching nor replication technique into the simulation. Nevertheless, note that SQS would benefit from data caching by placing the copies into exit points, so that the first query within a cluster C_1 would retrieve the results from the target cluster C_t and would store a copy of the results into the exit point at cluster C_1, namely $EP^Q_{C_1}$ for a query Q. However, any further query Q from another node from cluster C_1 would match the copy at exit point node $EP^Q_{C_1}$. This way, the solution would drastically reduce the response time, since the query would have a latency of $O(\log |C|)$ instead of a $O(\log N)$, where $|C|$ is the number of nodes at cluster C. This is a benefit of employing Cyclone as our underlying SPN.

4.1.5 Conclusions

In this work we have addressed the problem of a similarity query scheme for distributed peer-to-peer databases, namely SQS. We have proposed a latency-aware approach leveraging Cyclone SPN, that boosts the SQS Storage and Search Services, enabling similarity searches for multi-dimensional data spaces. As depicted in the API we provide along with SQS, we enable the coexistence of a wide variety of applications, where each of which can store different kinds of data, with no limitation on the data dimensionality. We have preferred to provide a generic range query system,

enhanced with global low communication cost, than to maintain data strictly contiguous as occurs in ZNet. To do so, SQS employs the *data adaptation module*, in particular the Bit Mapping function to adapt multi-dimensional objects to the SPN keyspace. In addition, SQS has based its range query algorithm on the generic range-based operation algorithm introduced along with the BM function in the data adaptation module (see Section 3.3). By doing so, our approach SQS has demonstrated its good qualities, such as the low communication cost and efficient query processing, all this through representative evaluations.

Summing up, with SQS we have presented the range query services of our framework. These services present the properties that we expect for any service being deployed in our framework, such as service efficiency, providing high-level (range query) services to a wide assortment of applications, as well as supporting their complex, multi-dimensional data spaces. In addition, by leveraging the data adaptation module in SQS, we have constructed a SPN-generic service. Therefore, we conclude with the fourth contribution of our thesis.

4.2 Geographical queries

Location-based services (LBS) receive world-wide attention as a consequence of the massive usage of mobile devices, but such location services require scalable distributed infrastructures in order to resolve spatial queries efficiently. We propose a novel methodology to enable geographical query support to distributed hash tables (DHTs), that was presented in our article [98]. To do so, we employ our framework and contribute with a new geographical information system service. In particular, the contributions of our methodology are the following ones: (a) our technique is SPN-generic, (b) it makes an effective clusterization of nodes and information into geographical areas, (c) providing data locality without sacrificing routing and data load balancing, (d) it is able to answer classical spatial range queries, as well as (e) a new kind of queries we call geocast, all of them in a distributed, scalable way. The feasibility and soundness of our approach are demonstrated through representative simulations.

4.2.1 Introduction

The importance of geographical information systems and location-based services has considerably increased in the last years [99]. This boom is a direct consequence of the proliferation of mobile devices, ubiquitous networking and positioning systems. Examples of such location-aware services include querying for specific resources in a geographic area or even integrating information collected by sensors in a given region [100], what are called geographical range queries. In this line, Google Local or Yahoo Local are centralized repositories that store location information and permit spatial queries using visual map interfaces. But we are interested in achieving a scalable, distributed solution.

It has been widely demonstrated that structured peer-to-peer systems (SPNs) [7, 8, 9, 13]) are suitable in the large scale. These systems perform lookups in an efficient *logarithmic cost* in terms of node hops, according to the number of nodes in the network. Moreover, SPNs provide *lookup correctness*, so that if a certain object *ob* exists in the system, it will be found. It is clearly a necessary property in modern (distributed) applications.

Nevertheless, supporting geographical range queries over structured peer-to-peer overlays is still an open research problem that implies strong challenges and a trade-off between them: *geographic information organization*, *multi-dimensional indexing*, *data locality*, and *data and routing load balancing*.

Geographical information organization. Geographical information is a clear example of (semantically) structured information. For instance, we can define that world geographical information is categorized within continent (disjoint) geographical areas; continent information is itself organized into country (disjoint) geographical areas; and so on until the necessary level of geographical precision is reached. Hence, we believe that a hierarchical substrate of clusters will greatly deal with the related complexity on hierarchical information organization in distributed systems. Thus, because we are interested in giving a generic solution to SPNs, instead of solving it for specific SPNs, *we introduce in this work a SPN-generic methodology to get geographical information efficiently clustered and, this way, perform efficiently spatial range queries over the system*. We have also designed a new kind of aggregated lookup so-called *geocast*, that *retrieves efficiently related information from the different local geographic levels* leveraging the clustered peer-to-peer substrate.

Multi-dimensional indexing. We assume a geographical information system as a *two-dimensional* problem solving: *Geographical location*, defining the geographical area where the information is related to, and *keywords*, describing the kind of resource made available. We generically call these two variables *geotags* and *tags*, respectively. $< geotag : Spain, tag : capital >: Madrid$ is an example of geographical data, formed by a pair of the form $< key >: value$. Thus, the system must perform in an efficient way insertions and searches of information by declaring both geotag and tag. To give an example, one can imagine a Google Local user looking for (tag:) *Italian restaurant*s in (geotag:) *Madrid*. And the user wants a delay-less answer with *related* results.

Data locality and data load balancing. Furthermore, as the uniform hashing functions employed in SPNs (e.g., SHA-1) destroy the semantic proximity, a *trade-off* is presented to the designer: how much data locality is preserved versus how much data load balancing is retained. If the adopted solution retains the data locality (e.g., by means of Space Filling Curves (SFCs) [33]), then there will not be data load balancing between nodes and, therefore, some external technique must be applied to balance the data load. Instead, if the solution ensures data load balancing (e.g., [31]), then this solution will not preserve the semantic proximity and performing higher level operations (like range queries) will be very expensive [55] in terms of communication cost.

Routing load balancing. This property enables the distributed system to route efficiently without hotspots. As conventional SPNs define the number of outgoing connections to other nodes, we measure the routing unbalancing in terms of the number of node incoming connections. To achieve routing load balancing, nodes should have the same number of incoming connections in average. Let us show a counterexample. Suppose a ring topology (like in Chord [7]) where most of the nodes are concentrated in a relative little ring sector, and few nodes are distributed throughout the rest big ring sector. In most of conventional SPNs, and in particular in Chord, it is completely an undesirable situation. Structurally such few nodes will be true hotspots, with a huge number of incoming connections and traffic. Instead, the concentrated nodes will receive very few incoming connections.

To the best of our knowledge, this work is the first in such a SPN-generic clustered proposal for an efficient organization of geographical information into a distributed peer-to-peer system, where we believe our work contributes in the following issues:

- We map the location information into the *suffix* of the node identifier (ID), leaving the ID prefix available for application-specific uses and providing at the same time *efficient routing* through the different geographical clusters (*geoclusters*). We call this suffix part the *clusterId*. Since our approach is based on the information mapping into the ID, regardless the specific SPN routing tables, we promote a SPN-generic approach.

- On the contrary to what could be initially expected, our approach provides *data locality* without sacrificing the overall node load balancing and SPN routing properties for skewed IDs. Furthermore, our system provides *data load balancing*, enhanced by the use of efficient cluster-based caching schemes that cannot be achieved in flat-based overlays.

- Our system supports efficiently *geographical range queries* combining both geotags and tags. Geotags are encoded in the *clusterId* and tags are stored in the appropriate cluster.

- Finally, we also present a novel search abstraction named *geocast query*. Inspired in the anycast proximity primitive, we enable *efficient local-to-global queries* (e.g., geocast tag:java) that returns information in increasing order of geographical area (specifically region, country, continent and world), leveraging the clustered peer-to-peer substrate and obtaining better performance results than in other flat approaches.

4.2.2 Related work

Geographical information systems have attracted researchers from very odd fields, included from the distributed system field. Moreover, we presented in Chapter 2 a review of peer-to-peer systems supporting spatial queries (Section 2.7). However, for the sake of clarity, we develop a little review in the following lines on the related work in the current field.

Geographical queries are in fact a special case of **multidimensional data** indexing, where existing structured peer-to-peer systems adopt any of the following two alternatives. One option is based on the use of multidimensional structures like R-trees and KD-trees like [101], but they do not properly scale for big networks. The second alternative consists in performing a reduction of the multidimensionality by means of linearization techniques, like Space Filling Curves (SFCs). SFCs have been intensively used in distributed geographical infrastructures, enabling the use of unidimensional structures like distributed hash tables (DHTs).

As data structures like R-trees do, the natural way of storing and managing spatial information is into a hierarchy. This way, *hierarchical peer-to-peer systems* [20, 42, 102, 103] arise to naturally provide this kind of geographical queries. In PlaceLab [102] and Brushwood [103] authors build a logical tree (Prefix Hash Tree (PHT) [55]) onto the DHT, but losing data locality in the system and, thus, resulting in very high costs of $O(\log^2(n))$ or $O(\log\log(n) \cdot \log(n))$ for structures with balanced depth. Some other work like Globase.KOM [20] build a hierarchical *superpeer*-based overlay. Nonetheless, a scheme based on superpeers forces some selected peers to have more responsibility, what is not always suitable [104].

Instead, Cyclone [42] is a technique for building *homogeneous* hierarchical DHTs [44] from their flat representation, and can be applied to existing structured peer-to-peer networks ([42] includes the Chord's case study). In this kind of hierarchical systems, there is no super-peer, but nodes participate in all levels they are present in (for further details see Section 4.2.3.2). Therefore, this approach makes no difference between peers, balancing the responsibility (and thus the load) through all peers.

Many authors have proposed to gain **data locality** using specific location-based node identifiers. Similarly in essence to IP subnetting, that partitions the IP address into network and node bits, in [105] authors embed location information in the ID prefix. This is however a risky approach, because not applying the consistent hashing for a random ID assignment will lead to non-uniform distribution of nodes and thus load unbalancing in the overlay. In the same line, TOPLUS [106], eCAN [23] and SCAN [107] achieve node and data locality taking advantage of IDs. Like before, in

a geographically uneven node distribution, these systems balance neither data nor routing load.

Overlays based on data-centric Skip Graphs/Nets [61] provide data locality while ensuring at the same time **routing load balancing**. SkipNet [26] achieves that using two IDs: a randomly selected *numericID* and a contiguous *nameID*. It is thus possible to use location-based IDs in Skipnet's nameID. In fact, we will compare our approach against SkipNet employing this *location-based nameID*, in order to take advantage of the SkipNet routing. The major drawback of such approaches is data load balancing: neither virtual node nor constrained load balancing techniques do really solve the aforementioned problem for skewed data sets.

Unfortunately, the above data-centric Skip Graph/Nets and tree-based approaches, which guarantee in some way data load balancing, leave unattended some of the challenges presented in this section: *geographic information organization*, *multi-dimensional indexing*, *data locality*, and *data and routing load balancing*. Furthermore, remember that our approach is SPN-generic and, thus, leverages the peer-to-peer substrate properties.

As for the term *geocast*, in mobile ad-hoc networks, geocast defines a geographically localized *multicast*. In other words, "geocast aims to send a message to some or all nodes within a geographic region." [108]. However, we see geocast as a geographically localized *lookup*, retrieving information from different geographical locations with a single operation.

4.2.3 Geophony: Geographical Information Services

We call Geophony to our geographical information services, that is included in our services framework. Geophony aims at providing geo-localization at the large scale. As we can see in Fig. 4.5, we use Cyclone [42] as the underlying SPN, over which our services rely on. In this case, we employ Symphony [13] as the instantiated SPN into Cyclone. We call Geophony to the resulting homogeneous hierarchical SPN, which by abuse of notation also names the geographical information services presented in this chapter.

Geophony services presented in the central tier are built by two main blocks: (i) the Storage Services that are responsible to record new information into the system, and (ii) the Search Services which provide *exact match* lookups, *spatial queries* and also our new primitive *geocast* search. Since the *store* function is very similar to the exact match query, we will only delve into the search services in the following sections.

As for adapting both data and queries to the underlying SPN keyspace, we base the new geographical information mapping approach on our aforementioned Bit Mapping services. In particular, we differentiate between the location mapping and the semantic information mapping when producing new IDs, so that these IDs get embedded both kinds of information and will benefit for node and data clusterization in terms of geographical proximity.

Lastly, in the top tier Applications are present, which employ our geographical information services in a distributed, scalable way. With Geophony services we aim at providing seamlessly geographical location services, boosting the query performance even in the presence of nodes non uniformly distributed into the SPN architecture.

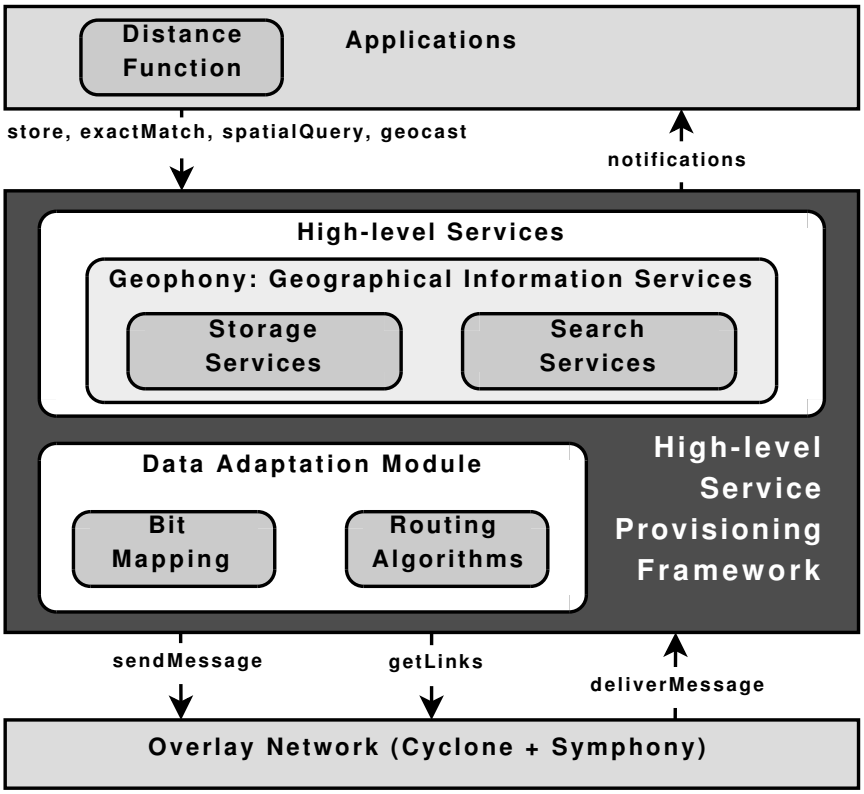


Figure 4.5: Geophony Architecture

4.2.3.1 The methodology

The idea behind our methodology consists in grouping nodes and information according to their geographical location. To do so, we define a hierarchy of disjoint geographical areas, which is traduced in a bit string and then embedded into the ID.

Doing so, not only does our methodology group nodes by their location, but also it guarantees that data is stored into the related geographical area.

Specifically, we propose to employ a *suffix-based assignment scheme to embed location information into the ID*. The novel contribution of our approach is that our design leverages the structural properties of the peer-to-peer substrate and, while embedding such location information into the suffix ID part (*clusterId*), our design supports geographical queries at distinct granularities, grouping nodes on successively more specific geographical areas, and sacrificing neither routing nor data load balancing.

We believe that homogeneous hierarchical peer-to-peer systems, like the provided by Cyclone [42], are necessary to deal with the current problem. This is why we have employed Cyclone into our approach. In particular, Cyclone nicely fits not only the hierarchical nature of a geographical information space, but also our suffix-based assignment scheme. As a case study, we detail how we apply the Cyclone algorithm onto Symphony [13] to produce what we call Geophony. We have selected Symphony as the Cyclone's instantiated SPN for its flexible selection of long links, that also enables an efficient routing in large scale networks with nodes non-uniformly distributed within the keyspace. Nevertheless, our approach is SPN-generic and, thus, *other SPNs* like Chord *can be used* instead of Symphony. We describe Geophony in the following section and how location information is embedded into the ID in Section 4.2.3.3.

4.2.3.2 A geographically clustered SPN

We apply Cyclone [42] algorithm to Symphony [13], producing a system we call Geophony [98], to reflect the hierarchical organization that naturally emerges in any geographical information system. Let us refer to Symphony as *conventional*, in clear reference to the *conventional* operation of the SPN which Cyclone is applied to. To explain Geophony, we describe in the following lines the clusterization mechanism, its routing table, routing algorithms and main properties. Even though we have already broadly introduced Cyclone in Section 2.1.2.1, we ellaborate on Geophony since it is part of our contributions.

Clusterization mechanism. As we have seen in Section 2.1.2.1, Cyclone is a generic technique to construct a homogeneous hierarchical SPN from a flat one, so that every node participates in all levels it pertains to (see Fig. 4.6a). This way, one can obtain the best of both worlds, without inheriting the disadvantages of either. This means that with the same number of links per node as in the flat SPN, the routing between any two nodes can be performed as efficiently as in the flat version. Its major virtue,

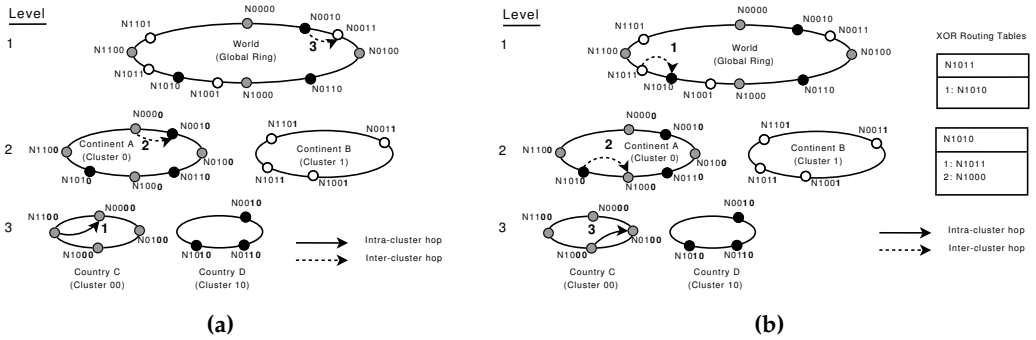


Figure 4.6: Geophony routing. **(a)** Hierarchy and (bottom-up) conventional routing example when node N1100 sends a message with key K0011. On every cluster, absolute greedy routing is performed, discarding the clusterId bits accordingly at each level. **(b)** Node N1011 needs to locate Country C (codified as 00) where to look for a location-based service (codified as 01). Steps 1 and 2 are part of (top-down) XOR routing in order to locate destination cluster. Step 3 is the part of conventional routing to locate the destination node (owner of key K0100).

however, lies in the utilization of IDs to represent real-world hierarchies. To explain how, suppose each node is assigned an ID from the range $[0, 2^b)$ so that the cl ($cl < b$) rightmost bits of its ID (expressed in binary form) are used to encode its *clusterId* at level l. Recursively applying this strategy in all levels, one can easily produce a hierarchy, where all nodes are organized into disjoint clusters at every level, having on level 1 a universal cluster with all nodes. Hence, each peer belongs simultaneously to at most η telescoping clusters (i.e., clusters of clusters of peers), with one in each level, where η denotes the hierarchy depth (e.g., depth 3 in Fig. 4.6a).

Routing table. In particular, Geophony is similar to Symphony. Each node n creates $O(\log N) + K$ links (for some little constant K and the number of nodes in the system N) to other nodes. The two operands $O(\log N)$ and K in the cost evaluation are necessary to reflect the two types of routing a geographical information system has to provide. We call the former (bottom-up) *conventional routing* and the latter (top-down) *XOR routing*, in clear reference to the way the routing is performed, respectively. The first $O(\log N)$ conventional links enables conventional SPN routing, forwarding queries to those nodes responsible of performing such operations; The last K XOR links are selected by the XOR metric and are necessary to locate a cluster of a certain geographical area (namely, geocluster) where to perform the user's operation. In other words, the localization of a cluster would be equivalent to zoom in/out

a map. Given that K is constant and that some of these links can coincide with those of $O(logN)$, we can consider that each node maintains $O(\log N)$ *different* links.

Routing algorithms. *Conventional routing* in Geophony is identical to routing in Symphony, namely, *absolute greedy routing*, but operating in loops. In the first loop, a node that wishes to route a query for a key k, it initially routes the message to the closest node p of k within its lowest cluster C' (i.e., p is the exit point node $EP^k_{C'}$). In the second loop, p switches to the next higher cluster and continues routing on that cluster. The same operation is repeated in each layer until the node responsible for k is reached. As the routing procedure goes up, more and more peers are included and the message is closer to the destination. At the last loop, routing is executed on the global cluster, which includes all peers in the system. It can be verified without much difficulty that Geophony achieves *logarithmic* routing when each node has degree $O(\log N)$ (as in Symphony). Fig. 4.6a depicts an example.

Note that this conventional routing enables to look up the responsible node of a query, but it is unable to locate a sibling cluster, in order to perform there some operation (e.g., a geographical query). *XOR routing* appears to deal with this challenge. XOR routing employs only the K XOR links and is similar to that in [15]. Node n routes greedily a query q to its i-th XOR node link p that minimizes the XOR distance between query q and node p, from all the XOR link set. This process is repeated until an arbitrary node m and query q share exactly the K rightmost bits. This means that the query has reached successfully the destination cluster. It is easy to demonstrate that with an amount of CL clusters, XOR routing reaches the destination cluster in $O(\log CL) \leq K$ node hops. Finally, conventional routing is employed to reach exactly the node responsible of query q within the destination cluster. The reader can observe in Fig. 4.6b an example of *XOR routing*.

Geophony properties. As noted in [42], a direct consequence of using suffixes as *clusterIds* is that Cyclone's performance is not affected by imbalances on clusters population, since Cyclone distributes uniformly the nodes within each cluster. Note that routing efficiency is based on the assumption that nodes are uniformly distributed along the keyspace $\mathcal{I} = [0, 2^b)$. To see why, consider that nodes in a lowest cluster C use the cl rightmost bits of their IDs to represent C's *clusterId*. Then, it is easy to see that by assigning the $b - cl$ leftmost bits of their IDs *uniformly at random*, these nodes become evenly distributed within C. In consequence, it results in routing load

balancing over the whole set of nodes, what is one of the important features of our approach.

Consistent with the previous discussion, Geophony provides another two key benefits. On the one hand, it makes sure that the path from a node to another never leaves the lowest cluster which contains both nodes, property known as *path locality*. On the other hand, it allows to store (resp. retrieve) information into (resp. from) a specific cluster representing a real geographical area, property known as *content locality*. Path locality provides *fault isolation and security*, since interactions between nodes within a cluster are not interfered with by node failures outside the cluster.

4.2.3.3 Location-based IDs over Geophony

As seen before, we propose to divide the ID into two fragments. The suffix part is the *clusterId* and identifies a cluster. The prefix part is the *nodeId* and determines a node within a cluster. It is easy to see that this ID structure nicely fits the Geophony hierarchy. Let us denote $|nodeId|$ and $|clusterId|$ as the precisions of these parts measured in number of bits, respectively.

Now, how $nodeId$ and $clusterId$ are defined for every node in the system? The $nodeId$ is drawn uniformly at random from the keyspace $[0, 2^{b-cl})$, thus guaranteeing a uniform load distribution within every cluster. The $clusterId$ gets a value representing its *geographical location*, given by the administrator in the joining process. Its bit size is flexible according to the application requirements. For instance, if we fix $|clusterId|$ to 64 bits, we will obtain a region precision of millimeters [99].

To put a detailed example, we define in this work a $clusterId$ based on political organization of nodes. To do so, we fix $|clusterId|$ to 40 bits and the $clusterId$ is partitioned in 3 geocluster levels, from right to left: *continent* (3 bits), *country* (7 bits) and *region* (30 bits) divisions; See Fig. 4.7 for this ID structure. Consequently, if one chooses 160-bit IDs like in Chord (i.e., $|nodeId| = 120$), every subarea supports up to 2^{120} nodes. We believe that these bit precisions are *necessary and sufficient* for the present and long time. Note that the region level guarantees a correct geographic precision, and that the ID partition cannot be dynamically configured because it must be consistent within all nodes. By obeying the partitioning to the political organization, we provide additional semantic information to the division of nodes into clusters. Complementary, the last *region* division enables the system to offer the necessary level of geographical precision to the solution. In addition, note that we place the most general concept (*continent*) into the rightmost part of the *clusterId*, and more specific concepts

(*country* and *region*) appear in its left side consecutively. This is not an arbitrary decision, but it enables the natural division of nodes and information into disjoint clusters. That is, in this case, nodes from different *region* clusters constitute a single *country* cluster, and in a similar way, several *country* clusters appear in the same *continent* cluster, where all *continent* clusters form a single *universal* cluster (see Fig. 4.6a).

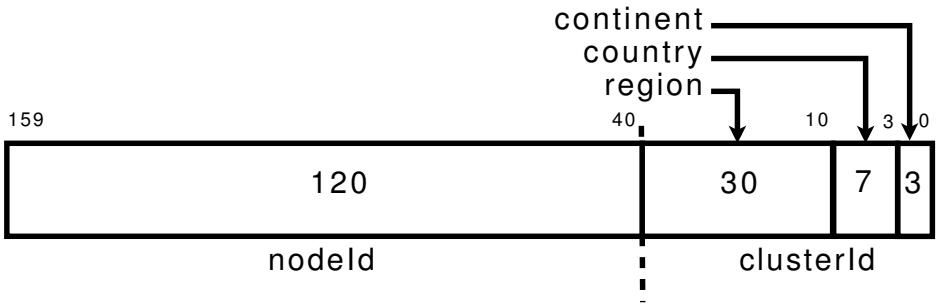


Figure 4.7: Node identifier (ID) structure reflecting the gegrophical division. The numbers inside the box tell the length in bits of the ID segment. The numbers over the box specify the position of the first bit (and last one) of each division within the ID.

We employ SFCs to guarantee a certain geographical contiguity. Therefore, this enables to perform queries for *local information* and *aggregated information* for all these geocluster levels. We set the codes for each geocluster level *recursively*. Thus, firstly, we apply the SFC on the $\{latitude, longitude\}$ centroid coordinates to set the code for the continent division; secondly the country division code and, lastly, the region code is set accordingly to the specific $\{latitude, longitude\}$ coordinates. For the simplicity of the algorithm, we have employed Z-curve [34] SFC, but other SFCs are also applicable (e.g., Hilbert Curve [35]). The region code is computed online by each node when joining or inserting data into the system. It is easy to see that all nodes belonging to different regions but to the same country, will appear merged in the country geocluster. Thus, we fix the number of XOR links K to be 10 (3+7), the number of necessary links to locate a country geocluster. Nonetheless, if we set $K = 40$, Geophony is able to address a region geocluster. Note that this ID construction is static (will not variate over the time) and independent of the peer-to-peer substrate instance, thus promoting a SPN-generic approach.

4.2.4 Routing and Data Load Balancing

As our methodology is SPN-generic, we do not evaluate how good is the specific SPN routing. Instead, we believe that we must analyse how the resulting homogeneous

hierarchical SPN can achieve routing and data load balancing and, moreover, given that actually information is not uniformly spread over the world.

In a ring-based overlay, the arc length defines both the incoming connectivity degree and the keyspace segment responsible for every node. Because nodes appear uniformly distributed at random within the leaf clusters, Geophony does not impose any responsibility unbalancing on the arc length and, in consequence, enables *routing and data load balancing*. Nevertheless, Geophony suffers from hotspots for skewed datasets in the same way that Symphony or other SPNs like Chord.

To almost mitigate the effects of this situation, *caching and replication techniques* appear in structured peer-to-peer systems. Given that queries for the same key always exit a cluster through the same node, which we call *exit point*, Geophony defines inherently a caching technique based on the query path. Thus, caching requested information on these nodes will mitigate drastically the query load on the responsible node. Moreover, within every cluster one can define a replication algorithm and, therefore, mitigate even more the incoming traffic of the responsible node. With both data caching and replication techniques, we believe that load balancing in the overall system is successfully addressed even for skewed datasets.

4.2.5 High-level queries

Our system provides three kind of queries: *exact match queries combining tag and geotag information*, *spatial range queries* over geographical coordinates and a novel proximity abstraction entitled *geocast query*. As explained before, geotags are encoded in the $clusterId$ and tags are stored accordingly in the cluster, i.e., the $nodeId$.

4.2.5.1 Exact match queries

Insertion (*put*) and search (*get*) of information in Geophony combining a geotag (lat,long, geocluster) and a tag (semantic information) simply implies calculating the appropriate ID. As explained before, the geotag is encoded using SFCs in the suffix, and the tag is encoded in the remaining prefix. As a consequence, the information will be stored in the cluster specified by the geotag (i.e., *clusterId*) and in the node responsible for storing the key (i.e., *nodeId*) inside that cluster. Therefore, we can efficiently store and retrieve *multidimensional data combining both tags and geotags*. Algorithm 4.2 details how this operation is performed by Geophony. For adapting tags into the $nodeId$ we use the lexicographic order (like in SkipNet). By doing this, we provide data locality

inside each cluster. It is obvious that skewed datasets can imply load balancing problems, but the cluster-based caching techniques presented in the previous section can almost overcome those problems.

Algorithm 4.2 *exact_match*
/* Directs the query combining the *tag* and *geotag* to the responsible node. */
Input: $n \leftarrow$ current node
Input: *geotag* $\leftarrow$ geocluster where search is directed to
Input: *tag* $\leftarrow$ keyword for filtering
Input: *level* $\leftarrow$ current geocluster level (default to *REGION*)
Input: *querying_node* $\leftarrow$ the query originator node

```
1:  key.prefix ← encode tag
2:  key.suffix ← encode geotag /* End BM adaptation */
3:  if n is responsible for key then /* n is the key's exit point at this cluster */
4:     if level ≠ WORLD then
5:        level ← higher(level)
6:        neighbors ← getLinks(n)
7:        next_node ← draw best node from neighbors according to the conventional
          routing for level level
8:        exact_match(next_node, geotag, tag, level, querying_node)
9:     end if
10:    result_set ← {o ∈ O' | M_Q({geotag, tag}, o) = 0} /* n is the responsible node
       for key */
11:    send(n, result_set, querying_node)
12: else /* Forwarding the query into the actual cluster */
13:    neighbors ← getLinks(n)
14:    next_node ← draw best node from neighbors according to the conventional rout-
       ing for level level
15:    exact_match(next_node, geotag, tag, level, querying_node)
16: end if
```

In Algorithm 4.2 is depicted how the exact match query is processed by nodes along the query path. The idea behind this algorithm is to route the query in loops (line 4) along all participating clusters as Geophony establishes. When the query is forwarded at the immediate higher level (line 8), the node n is routing up to the next higher, geographically wider cluster. Instead, the forwarding process of line 15 is due to a routing within the current cluster. Note that only exit point nodes execute lines 3-9, while the responsible node for key *key* executes the lines 10-11. In other words, this last node n will provide directly to the *querying_node* the results matching

to the exact match query. When changing of geographical precision, we designed the function $higher(level)$ that returns just the immediate higher geographical level accordingly, i.e., $COUNTRY \leftarrow higher(REGION)$, and so on, where the possible results are *COUNTRY*, *CONTINENT* and eventually *WORLD*.

Searching for information like <*geotag:lat,long, tag:jazz*> involves a simple lookup operation in the overlay with *optimal logarithmic routing cost*. Additionally, we can also perform queries using *wildcards* like in <*geotag:lat, long, tag:**> and <*geotag:**, *tag:jazz*> at the same cost. Specifically, the first wildcard enables the user to retrieve any sort of information existing in a certain place (geocluster). The second wildcard enables to retrieve any information related to *jazz* stored at the global ring.

4.2.5.2 Spatial range queries

A spatial range query is understood as a search of any kind of information located in a specific geographical area. For simplicity[1], we describe this area by means of the rectangular region defined by *(latMin, longMin)* and *(latMax, longMax)*. Traditional spatial databases store geographical data using hierarchical structures (R-tree, KD-tree) by defining Minimum Bounding Rectangles (MBRs). Leaf nodes in the R-tree contain entries of the form $(data, mbr)$ and non-leaf nodes contain entries of the form $(ptr, rect)$, where *ptr* is a pointer to a child node in the R-tree and *rect* is the MBR that covers all the MBRs in the child node. Unfortunately, adapting these algorithms to a distributed way is a very complex problem in the big scale.

Like in PlaceLab [102], our system can be seen also as a trie-based topology that partitions the space and thus permits to have implicit knowledge about key locations in the hierarchy. This clearly fits the SPN lookup mechanisms and it avoids the fragility of distributed tree topologies.

Geophony provides enhanced spatial range queries. In addition to the traditional ones, our system enables to combine the query with tags (keywords) to further filter search information on the system side, instead of on the user side. To achieve that, the spatial range query is defined as follows. We must calculate the *linearized suffix* that minimally encompasses the entire query zone. Using Z-curve, we first obtain the longest common prefix of minimum and maximum ranges for this query, i.e., $(zMin, zMax)$. Then we invert this prefix and we encode it in the *clusterId*. Now, we can use this suffix to traverse the tree until we reach the responsible node(s) of the specified bounding rectangle, based on our Algorithm 3.1 for range-based operations.

[1] In cases when applications use the pair Global Positioning System (GPS) coordinate and a radius, it can be easily provided as a rectangular region, by calculating its minimum bounding rectangle.

In this situation, a total of $O(\log CL + \log |(zMin, zMax)|)$ hops are needed to deliver the query to the responsible node(s), where CL is the total number of clusters and $|(zMin, zMax)|$ is the number of nodes into the search area of the responsible cluster C_i. Note the upper bound $|(zMin, zMax)| \leq |C_i|$, when search area overlaps the whole geocluster area. We forward the query through Geophony XOR routing to locate the target cluster, employing $O(\log CL)$ number of hops to realize such an operation. Afterwards, conventional routing delivers the query to the responsible node(s) in that cluster with $O(\log |(zMin, zMax)|)$ hops. Algorithms 4.3 and 4.4 details how this operation is performed, and Fig. 4.6b depicts an example of this combination of routing schemes.

Algorithm 4.3 *spatial_range_query*
/* Starts the operation, directing the query to the requested geocluster. */
Input: $n \leftarrow$ the query originator node
Input: $locMin, locMax \leftarrow$ define the area covered by the query
Input: $zMin, zMax \leftarrow$ mapped area covered by the query, calculated as:
$zMin \leftarrow Zcode(locMin.lat, locMin.lng)$
$zMax \leftarrow Zcode(locMax.lat, locMax.lng)$
Input: $tag \leftarrow$ keyword for filtering purposes
Input: $querying\_node \leftarrow$ the query originator node

1: $clusterId \leftarrow longest\_common\_prefix(zMin, zMax)$
2: **if** $n.clusterId = clusterId$ **then**
3: $process\_spatial\_query(n, (locMin, locMax), (zMin, zMax), tag, querying\_node)$
4: **else** /* XOR routing to reach target cluster */
5: $XORneighbors \leftarrow getLinks_{XOR}(node)$
6: $next\_node \leftarrow \exists q \in XORneighbors : q.clusterId = clusterId,$
$\forall p \in XORneighbors \setminus \{q\},$
$\mathcal{M}_{\mathcal{P}}^{XOR}(q.clusterId, clusterId) < \mathcal{M}_{\mathcal{P}}^{XOR}(p.clusterId, clusterId)$
7: $spatial\_range\_query(next\_node, (locMin, locMax), (zMin, zMax), tag,$
$querying\_node)$
8: **end if**

This Algorithm 4.3 performs the XOR routing to reach the target cluster where to realize the spatial query. The searched area covers the region within the rectangle defined by the two points $(locMin.lat, locMin.lng)$ and $(locMax.lat, locMax.lng)$. This information is encoded using the $Zcode()$ function, that calculates the Z-code according to the Z-curve linearization function. When a node pertaining to the target cluster is reached (line 2), this makes to start the spatial query within that cluster (line 3). See Algorithm 4.4 to see how the spatial query is eventually developed. Conversely, the

query is forwarded using only the XOR routing table (lines 4-8). As outlined before, we use the *greedy XOR routing* similar to Kademlia one [15]. Line 6 depicts the selection of the XOR link that is closer numerically to the target cluster, where $\mathcal{M}_{\mathcal{P}}^{XOR}(a,b)$ evaluates the XOR distance between clusterIds a and b.

Algorithm 4.4 *process_spatial_query*
/* Parallelize the spatial range query for those nodes responsible of a (sub)area of the query. */
Input: $n \leftarrow$ current node, responsible of a part of the area
Input: $locMin, locMax \leftarrow$ total area covered by the query
Input: $zMin, zMax \leftarrow$ mapped (sub)area covered by the (sub)query
Input: $tag \leftarrow$ keyword for filtering
Input: $querying\_node \leftarrow$ the query originator node

1: $localKS \leftarrow clusterId(segment(n,n)) \cap [zMin..zMax]$
2: **if** $localKS \neq \emptyset$ **then**
3: $\quad result\_set \leftarrow \{o \in \mathcal{O}' \mid \mathcal{M}_{\mathcal{Q}}(\{(locMin, locMax), tag\}, o) = 0\}$
4: $\quad send(n, result\_set, querying\_node)$
5: **end if**
6: $node\_set \leftarrow \{node \in getLinks(n) \mid node.clusterId \in [zMin..zMax]\}$
7: **for all** *node* in *node_set* **do**
8: $\quad nodeKS \leftarrow clusterId(segment(n, node)) \cap [zMin..zMax]$
9: $\quad process\_spatial\_query(node, (locMin, locMax), (nodeKS.min, nodeKS.max), tag, querying\_node)$
10: **end for**

Algorithm 4.4 concludes the spatial query by visiting all nodes that are responsible for the area $(locMin, locMax)$. All nodes responsible for a query's subarea (lines 2-5) retrieve all objects matching both the spatial as well as the tag premises (line 3). In addition, every participating node will forward the query to its neighbors that potentially are responsible for part of the covered area (lines 6-10). Since the covered area for every forwarding node is calculated and adapted (line 8), this algorithm ensures that nodes will be visited only once and that the whole area covered by the spatial query is traversed.

Summing up, we take advantage of suffixes (i.e., *clusterId* in Geophony) to efficiently locate *multidimensional data* in specific clusters. In addition, we combine these spatial range queries with *semantic keywords* (tags) to further filter the results in the system side.

4.2.5.3 Geocast queries

The *geocast query* presented in this work is a new primitive which permits a user to recover information associated with an arbitrary tag (in a similar way to the anycast in [109]) in all user's geoclusters very efficiently: starting from user's region, continuing on user's country and so on until the global owner of the tag is found.

For example, consider a Spanish SUN researcher that wishes to retrieve the information about the projects in which it participates at all scales. That is, the SUN projects that are being developed in Spain, the European ones, and finally the worldwide SUN projects. Using a geocast query for the tag SUN, our system guarantees that all the above information can be recovered within $O(\log N)$ routing hops, in stark contrast to a conventional SPN, in which typically four queries of $O(\log N)$ hops each would be required (informally, one query for each pair <*geotag:region, tag:SUN*>, <*geotag:country, tag:SUN*>, <*geotag:continent, tag:SUN*> and <*geotag:world, tag:SUN*>).

Algorithm 4.5 *geocast*
/* Recursively, nodes at geocluster's exit points for the given *tag* answer to the *geocast* query. The operation starts with the following call:
geocast(querying_node, tag, REGION, querying_node) */
Input: $n \leftarrow$ current node
Input: $tag \leftarrow$ searched keyword in geocast
Input: $level \leftarrow$ current geocluster level (default to $REGION$)
Input: $querying\_node \leftarrow$ the query originator node

1: **if** n is responsible for tag at level $level$ **then** /* n is the EP^{tag}_{level} */
2: $result\_set \leftarrow local\_search(n, tag, level)$
3: **if** $result\_set \neq \emptyset$ **then**
4: $send(n, result\_set, querying\_node)$ /* direct sending */
5: **end if**
6: **if** $level \neq WORLD$ **then**
7: $geocast(n, tag, higher(level), querying\_node)$
8: **end if**
9: **else**
10: $neighbors \leftarrow geLinks(n)$
11: $next\_node \leftarrow$ draw best node from *neighbors* according to the conventional routing for level *level*
12: $geocast(next\_node, tag, level, querying\_node)$
13: **end if**

As described in Alg. 4.5, the idea is to iterate the geographical hierarchy, routing

the geocast query for tag T using the *conventional routing*, starting from the lowest geocluster in which the querying node lives. This is done by setting the level to the *REGION* level. This way, the geocast algorithm benefits from the hierarchical greedy routing described above. See Fig. 4.6a for an example.

Once the node p_T responsible for T in the regional geocluster C_{REGION} is found in line `1` (i.e., the exit point $EP^T_{C_{REGION}}$ for tag T in this cluster), p_T returns the result for the pair <*geotag:region, tag:T*> if any (lines `3-5`), and continues routing on the next higher geocluster (lines `6-8`), known as country geocluster. This operation is repeated for each geocluster until the owner for T is reached in the world geocluster. Conversely, whenever a node p is visited and is not responsible for the tag T at the current level *level*, node p just forwards the query according to the conventional routing algorithm (lines `9-13`).

Our system guarantees that all the above information can be recovered within $O(\log N)$ routing hops, in stark contrast to a conventional flat SPN, in which typically four queries of $O(\log N)$ hops each would be required, one for every geocluster level. To see effectively how this algorithm is processed, Fig. 4.6a depicts an example. In this case, *N1100* is the querying node, *N0000* the country exit point, *N0010* the continent exit point and *N0011* the world owner node. Furthermore, after every step an answer to node *N1100* is sent. In particular, we view our geocast abstraction as a typical $get()$ operation, which is optimal in number of routing hops ($O(\log N)$ routing hops on average). To the best of our knowledge, we do not know of any other system that offers such functionality as efficiently as our approach.

4.2.6 Evaluation

We present here some simulation results to validate and illustrate the contributions we achieve with our methodology. As we have seen, Skip Graphs/Nets have been having a lot of interest lastly for their properties, capabilities and above all for providing nicely range queries. For this reason, we compare Geophony against SkipNet [26].

To make a fair comparison, we assume that in both systems each node maintains $O(\log N)$ neighbours as we discussed above (in this scenario the other K XOR links are useless). For simplicity in the simulation scenario, we choose $b = 24$, so IDs are binary strings of 24 bits. In addition, we varied the number of levels from 1 (Symphony) to 4, with $K = |clusterId|$ of 0, 3, 5 and 7 bits, respectively. We also assume a Normal distribution assignment of each node to any of the 2^K geoclusters, with $\sigma = 0.125$ and a distributed uniformly at random μ. The reason why we use such skewed distribution

is to emphasize the fact that Geophony, similar to that happens in SkipNet, is insensitive to the local distribution of nodes in each cluster. We vary the number of nodes between 1K and 20K, and, for each network size, we run at least 10 differently seeded experiments, consisting of *10K random requests* each.

For the sake of clarity, we will refer as *latency* to the number of visited nodes by the execution of a distributed query. When necessary, we will specify the kind of nodes that the latency property refers to.

4.2.6.1 Routing and Data Load Balancing

Let us first evaluate numerically the average latency (in terms of number of routing hops) of Geophony vs. Skipnet (Fig. 4.8a). Note that a lower average latency means lower network delays experienced by an end user that frequently performs exact match queries. To that effect, in Fig. 4.8a we plot the routing latency averaged over all nodes and all requests. Although the number of links per node is $O(\log N)$ for both geometries, the figure shows that Geophony provides a lower average latency than SkipNet, irrespective of the number of levels in the hierarchy. However, it is important to note here that the average latency increases *slightly* when the number of levels in the hierarchy increases. We note, besides, that this increase is at most 2 hops. The reason for this increase lies in Geophony hierarchical routing. While in Symphony every node has all links available to route a query, Geophony forces a query to cross the exit points at each level, inducing a path that is not always the optimal route (notice that a node does not have available all links until level 1). In addition, although the number of hops increases, these hops should be faster than hops in the flat SPN, since they are inside a cluster of *potentially* closer nodes.

We next evaluate the performance of both systems assuming that all nodes are able to cache answers (Fig. 4.8b, 4.8c and 4.8d). We assume that both geometries use *path caching* that consists in caching the answer on all nodes through which the query is routed. We use path caching since it is considered the most deployed caching technique for overlay networks [110]. Although at first glance it may seem inappropriate to use caching as metric to compare both systems, we argue that it is important to measure the potential reduction in latency a system can experience in the face of reiterative queries. We point out that this evaluation is somewhat complementary to that in Fig. 4.8a, but remarking the fact that a hierarchical substrate is ideal to handle geographical information. Our motivation stems from the observation that caching, and more generally, content delivery networks, are one of the most deployed applications of network overlays.

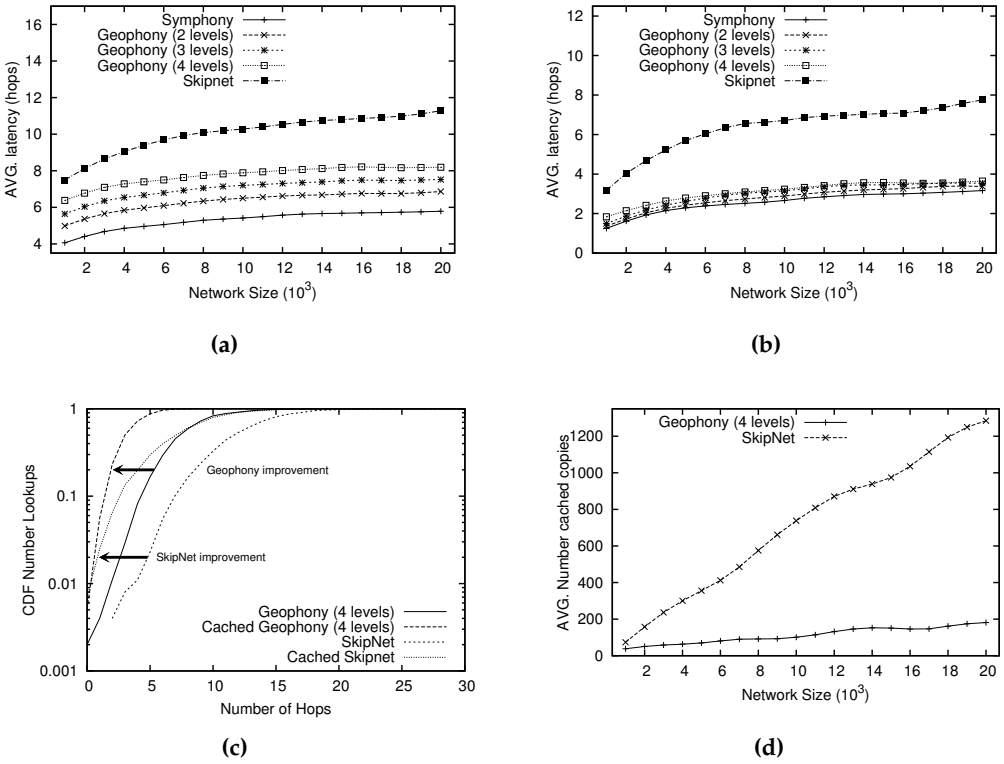


Figure 4.8: Geophony vs SkipNet exact match query evaluation. **(a)** Routing hops without caching. **(b)** Routing hops with at most 128 cached answers. **(c)** Caching effect evaluation (4.8b vs 4.8a). **(d)** Number of caching copies, fixing lookups to an average of 4 hops from all nodes within the network.

Firstly, we perform the following two simulations. In the first simulation, we evaluate the latency reduction that the geometries experience when the maximum number of answers to be cached for a given query is set to 128 (Fig. 4.8b). Geophony, which takes advantage of exit points, experiences a reduction in latency greater than in SkipNet, which stores answers at nodes that might not be on the route to the destination. We can see also from Fig. 4.8c that, on the existence of caching in both systems, Geophony achieves clearly a reduction of the number of hops per query (almost the 95% in 4 hops).

In the second simulation, taking the opposite view, we investigate how many caching copies are required to store an answer A, so that the average latency to access A does not exceed a total amount of 4 hops (Fig. 4.8d). To that effect, we assume that all nodes perform the same query. For the sake of clarity, in this plot we only include the results corresponding to a 4-level Geophony, which is the worst setup for this experiment (recall that the average latency increases with the hierarchy depth). As

expected, SkipNet requires to cache more answers than Geophony to maintain an expected latency of just 4 routing hops. Also, notice that the number of cached answers remains nearly constant, irrespective of the number of nodes in the system, thus proving the intuition that with our methodology and, in particular, *using exit points nodes* as rendezvous where to store data caching copies *the efficacy of caching is greater*.

4.2.6.2 Spatial and Geocast queries

As SkipNet is a data-centric Skip Graph-based overlay, where range queries are performed efficiently, we foresee that both systems will perform in a similar operation cost on spatial range queries. On the other hand, we foresee that Geophony will clearly outperform SkipNet in the geocast evaluation. We employ the same mapping mechanism on both systems, but in SkipNet it appears as a prefix-based mapping, as we have announced in Section 4.2.2, in order to take advantage of the SkipNet routing. We evaluate the performance of both kind of queries by the average latency.

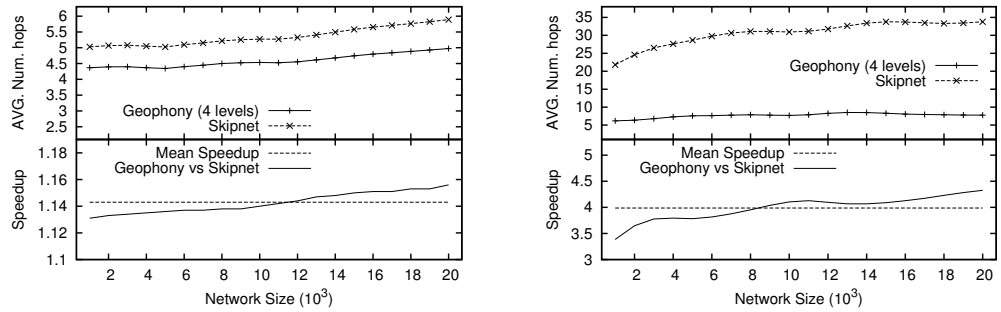


Figure 4.9: Geophony vs SkipNet evaluation. **(a)** Spatial range queries: Number of hops and improvement. **(b)** Geocast queries: Number of hops and improvement.

Since SkipNet was designed to enable nicely range queries with a logarithmic cost, both systems result nearly equivalent in spatial range queries, with a little improvement ratio from 12% to 14% of Geophony vs. SkipNet (see Fig. 4.9a). Moreover, by means of SkipNet's double routing (by nameID and numbericID), SkipNet and Geophony have routing and data load balancing.

Conversely, as it can be seen from Fig. 4.9b, in the geocast query evaluation our expectations come true. Geophony improves nearly 3 times in average over SkipNet (namely, SkipNet number of hops are 4 times greater than those of Geophony). It is easy to see that geocast queries in Geophony nicely fit the hierarchical overlay structure, but SkipNet, a flat data-centric SPN, is penalized by performing as many queries

as geographical levels appear embedded into the ID (4 in our case: region, country, continent and world).

4.2.7 Conclusions

With this work we have introduced our fifth contribution, a novel SPN-generic methodology to support geographical queries onto existing SPNs. This work is based on the results published in our article [98]. Using the Cyclone algorithm, we have created Geophony, a hierarchical version of Symphony. We propose a novel suffix-based location ID assignment in order to map geographical areas (i.e., continent, country and region) and coordinates (using SFCs) to the suffix ID (i.e., *clusterId*). On the contrary to what could be initially expected, our approach provides data locality without sacrificing the overall routing and data load balancing properties for skewed IDs. Furthermore, our system supports spatial range queries combining location information (geotags) and semantic keywords (tags). We also have presented the geocast search abstraction, enabling efficient local incremental queries over geoclusters. We have provided clear validation results that demonstrate that our approach outperforms flat data-centric overlays (SkipNet) in data load balancing and geocast queries. This is a consequence of the hierarchical clustered architecture of our model.

4.3 Summary

In this chapter we have presented two out of our contributions, which constitute together the two module services of our **data management module**. First, the SQS similarity services are presented in Section 4.1. SQS represents the module service capable of performing multi-dimensional range queries. Second, Geophony services follow in Section 4.2, which forms our proposal to tackle the geo-localization necessities of geographical information systems.

In this case, both module services rely on hierarchical SPNs based on Cyclone [42], with different instantiated SPNs in each (Chord in the case of SQS and Symphony in Geophony). Even though we use Cyclone as the methodology to construct hierarchical versions of flat SPNs, the resulting hierarchical SPNs have the proper characteristics of the instantiated SPN (Chord and Symphony, respectively). Therefore, we proved that our framework can be successfully deployed onto different SPN infrastructures.

Consistent with our initial design, the two module services utilize the services provided by our framework and, in particular, our Bit Mapping techniques to adapt data to the specific SPN keyspace. In Geophony services, though, we apply a smooth

variation to the standard mapping function, differentiating between the location information (embedded into the suffix part) and the semantic information (embedded into the prefix part).

Indeed, we addressed a fair evaluation of our module services against other works, where we have proven the feasibility, soundness and efficiency of our approaches, improving the performance of existing solutions. In addition, we presented new primitives such as the *geocast* query, which allows users to retrieve information from several geographical granularities at once. As for geocast queries, we proved that Geophony clearly outperforms to flat SPN architectures when processing geocast searches, due to the hierarchical nature of the query as well as the Geophony's node structure.

In the future, top-k queries and other searches based on data aggregation could be developed as new module services in our framework, taking advantage of all the existing services.

5

Content distribution capabilities

This chapter introduces the **content distribution module**, which leverage our services framework, and in particular our adaptation technique, to provide content-based publish/subscribe services. Content-based publish/subscribe services are more complex to design, develop and deploy than those topic-based ones, given that users can establish predicates over the data flowing into the information system, to filter out the information that users are not interested in. The undermentioned publish/subscribe module service is based on the work published in [111].

5.1 Introduction

Publish/subscribe (pub/sub) systems are powerful mechanisms for information dissemination. These systems are characterized by two main actors. *Publishers* are those actors who produce information either periodically or sporadicly. Usually, the literature denotes such pieces of information as *events*. Additionally, *subscribers* are those actors that are interested in receiving significant events. They employ *subscriptions* to define their particular interests, expressing conditions on the content of events (*content-based* model) or just on a category they belong to (*topic-based* model).

The aim of these systems is to deliver all events from publishers to corresponding subscribers. To do so, most of the existing pub/sub systems build a specific pub/sub peer-to-peer overlay [22, 76, 77, 84], also called event brokering networks. In some other cases, these solutions are constructed onto other overlay networks, called distributed hash tables (DHTs) [7, 8, 9, 11], like Scribe [21], Bayeoux [79] or PastryStrings [83]. This kind of approximations incur additional management costs, concerning the pub/sub overlay, given that the underlaying SPN also performs the same tasks. The tasks we are referring to are related to the overlay maintenance; for instance, setting up recently joined nodes or guaranteeing the correct overlay connectivity, even in the presence of node failures.

SPNs were introduced as scalable data structures for building large distributed applications. Peter Triantafillou and Ioannis Aekaterinidis in [80] introduced one

of the first content-based approximations where Chord [7] is employed as reliable routing infrastructure. Therefore, they do not build a specific pub/sub overlay. To do so, they employ the *rendezvous model*. The motivation behind that is because the multihop routing abstraction implemented by SPNs integrates naturally with the need for globally unique rendezvous nodes in these rendezvous-based routing approaches. Nevertheless, when these approaches perform a SPN communication *for each event's attribute mapping* (as applied in [80]), such systems suffer lack of scalability on high-dimensional contexts.

Later, [82] perform a similar approach but, in this case, they perform a particular mapping of events and subscriptions to keys from the SPN keyspace, instead of per-attribute mappings. Even though their approximation is quite interesting, their system requires some sort of *primitive multicast* provided by the SPN in order to obtain good performance.

In summary, the design of event dissemination in content-based pub/sub systems working onto SPNs has to take into account different factors:

- *Lighweight and portable proposal.* The goal behind the idea of building the pub/sub system onto SPNs is twofold: (i) leverage the SPN routing infrastructure and, thus, avoiding to build a pub/sub overlay protocol over an existing SPN, and (ii) operate suitably onto (most of) current SPNs, without requiring ad-hoc functionalities to the SPN. Nevertheless, important properties like load balancing and low local state information maintenance must be retained.

- *Multiple sources.* Nodes cooperating in this distributed pub/sub system should have guarantees for publishing and subscribing at any time and concurrently.

- *Multi-attribute data.* Usually content-based systems support applications whose information is defined in terms of different parameters or attributes. Therefore, events contain a value per attribute, and subscriptions specify range of values of interest for each attribute. In consequence, the system needs some mechanism to route multi-dimensional events and subscriptions throughout the network, while SPNs operates with one-dimensional keyspaces.

As seen before, current systems (e.g., [80, 82]) do not deal with all the factors above described. For this reason, we introduce in this work a novel system, called CAPS, that builds *content-based event dissemination infrastructures* onto SPNs in a scalable, efficient way. To do so, we employ the *rendezvous model* in order to meet both events and subscriptions. For this reason, the system defines a certain set of nodes from the SPN

as *rendezvous nodes*, being responsible of matching events against subscriptions and start then the notification process. Additionally, these rendezvous nodes are selected deterministically, so that the node in the SPN responsible for a given key then becomes the rendezvous node. Due to the SPN properties, the chosen node will be globally agreed upon by all nodes and, this way, every node can use the peer-to-peer routing substrate to send messages to this rendezvous node.

In summary, this work introduces the following contributions that will be discussed along this chapter:

- The *rendezvous model* enables the system to avoid the construction of a specific overlay to disseminate events in a proper way. In fact, CAPS leverages the SPN routing properties to set rendezvous nodes every time, therefore achieving a **lightweight pub/sub system**. Unlike other pub/sub systems, CAPS does not need advertisements to meet both events with subscriptions, proportioning even a more lightweight solution. We also design *SPN-generic* subscription and notification algorithms that allow CAPS to work onto different SPNs, making the whole solution **portable**.

- CAPS employs a hash function to map every *subscription* into *a set of keys* and every *event* into *a key*, deterministically, in order to deal naturally with **multi-dimensional domains**, and **multiple sources** cooperating within the system.

- We realize a **complete analysis** both theoretical (seen in Section 3.3.3) and experimental (later in this chapter) on *high-dimensional* scenarios and with a *wide range of selectivity ratios*, which define the ranges of interest of subscriptions against events. This evaluation demonstrates that CAPS has better performance on high-dimensional contexts, requiring low memory capacity and hops to perform event disseminations and subscriptions. To the best of our knowledge, this is the first formal performance study of this kind of event dissemination based on the rendezvous model working onto SPNs.

The system is presented by introducing (i) a completed related work (previously in this section, as well as in Section 2.8), (ii) a deeper formal analysis, formerly in Section 3.3.3, (iii) the design of our proposal in Section 5.2, and (iv) the performance evaluation through simulation on relevant scenarios in Section 5.3.

5.2 The CAPS System

In this section we describe the CAPS architecture and operation, including subscription setup and event dissemination.

5.2.1 System Overview

The CAPS goal is to provide content-based event dissemination from publishers to subscribers. From an architectural viewpoint, CAPS provides this functionality to Applications in the upper layer and, to do so, the system is settled down into our framework, as we can see in Fig. 5.1.

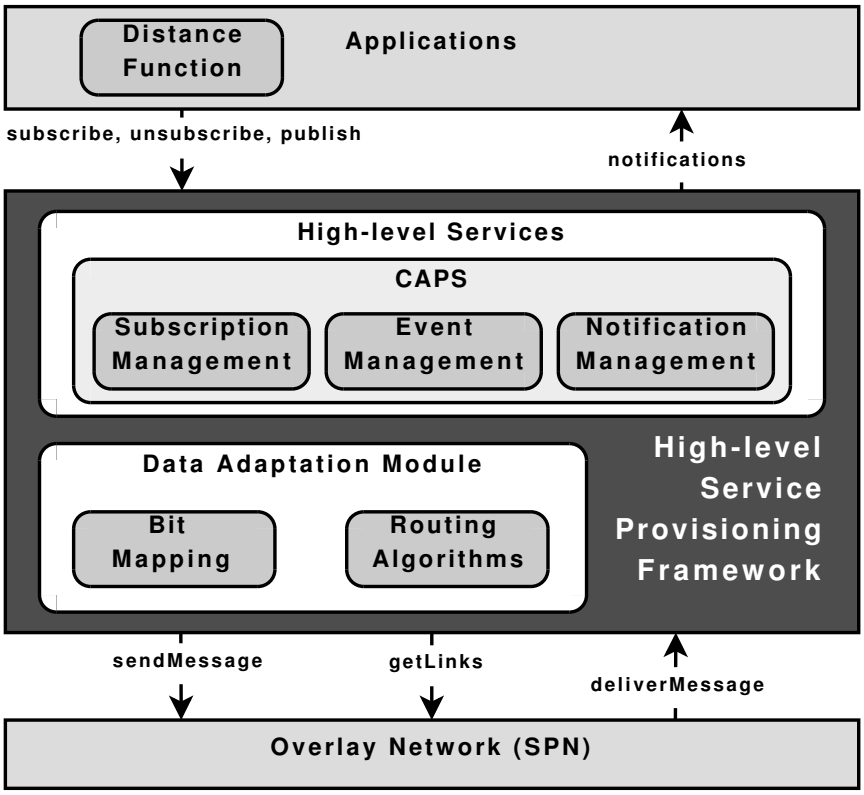


Figure 5.1: CAPS system components and context.

In this context, Applications can behave as publishers and subscribers at the same time, and CAPS will notify them with all events that match their particular interests. By employing CAPS into our framework, CAPS benefits from the **data adaptation module**, and in particular from the BM function, to adapt events and subscriptions

to the SPN keyspace. In addition, the provided range-based routing algorithms will contribute in maintaining the system consistency, as we will detail later in this chapter.

As before, our framework employs a SPN as its routing infrastructure, leveraging the proper SPN's properties. In this line, CAPS will use the three basic functions that SPNs provide to our framework as follows. *sendMessage* enables the system to send a CAPS message employing the SPN infrastructure. The SPN node provides to the framework (as well as to CAPS) the whole set of links from its local routing table, when is called the *getLinks* function. Finally, *deliverMessage* function is called when the SPN node has to deliver a message to our service. Given that our framework employs an arbitrary SPN as routing substrate, the system design and analysis are focused on the subscription and event notification management.

As CAPS is based on the rendezvous model, the system employs a hash function in order to map subscriptions and events into keys within the SPN keyspace. This way, CAPS determines the rendezvous nodes as the owner nodes of the given keys. All these operations together with the subscription installation and event notification appear defined in the following functional blocks.

The first we find is the *data adaptation module* (see Fig. 5.1). The transformation of multi-dimensional events and subscriptions into one or more keys is described in Section 3.3, taking events as *objects* and subscriptions as *range objects*, respectively. These keys allow the message routing through the SPN transparently. *Subscription management* block realizes the Applications' subscription and unsubscription tasks. Any subscription and unsubscription is packed into a message, that is delivered to the SPN infrastructure according to the CAPS subscription's routing algorithm and the keys obtained from the above mapping. On the other hand, when one particular node becomes rendezvous for a given subscription, this block also stores it locally. *Event management* block receives recently published events from the Applications and delivers them to the rendezvous nodes individually, employing the SPN infrastructure. Finally, *notification management* block is responsible in rendezvous nodes of matching events against the locally stored subscriptions and to start then the notification process. Additionally, when necessary, this block locally notifies Applications of events matching their interests.

5.2.2 System Implementation

Given that Applications operate with multi-dimensional data and the SPN keyspace is one-dimensional, CAPS defines a dimensional reduction operation in order to deliver

events and subscriptions to rendezvous nodes. This section details the design of the mapping procedure and the routing algorithms.

5.2.2.1 Subscription Management

Intuitively, the key idea behind the subscription management based on the rendezvous model is to install subscriptions in those nodes who will be the rendezvous nodes for future events matching these subscriptions. Thus, when Applications perform subscriptions, CAPS has to install the given subscriptions to rendezvous nodes.

There are two subscription/unsubscription schemes available: (i) **stateful subscription scheme** in what a subscription will remain in the system until it is removed (unsubscribed) or the subscriber fails, and (ii) **stateless subscription scheme** in what subscriptions are removed after a lease and, therefore, they must be re-subscribed to continue present in the system. One of the most important goals of such an election is to reduce the overall subscription and unsubscription communication, always maintaining up-to-date subscriptions. Thus, we have chosen the stateful approach because we consider that nodes are quite stable. If nodes are relatively unstable, the stateless approach is recommended. We detail the subscription procedure by the following steps: subscription mapping, installation, storage and unsubscription.

Subscription mapping. When an Application performs a subscription, this take the form $\{[min_1..max_1], [min_2..max_2], .., [min_D..max_D]\}$, where every $[min_i..max_i], i = 1, 2, .., D$, defines the subscriber's interest for the *i-th* dimension. Such interests may coincide with the entire dimension domain, but also it can occur that $min_i = max_i$, for an arbitrary *i-th* dimension. In the last case, the subscriber is only interested in only a certain value. The result of this operation is a set of keys as described in Section 3.3.1.2.

Subscription installation. The subscriber CAPS node, subscriber for the rest of the chapter, is now ready to install the subscription into the system. Briefly, the subscriber starts the process employing the Algorithm *subscribe* (see Alg. 5.1) and ends when all rendezvous nodes of keys within the keyset KS store the subscription. Subscriptions are sent together with the subscriber ID, in order to be notified when necessary. Note that even when $|KS| > 1$, only one node can be the owner of the entire keyset.

Instead of building the whole set of keys (note that can be a considerable amount of keys and thus unscalable), we have defined an algorithm that builds the necessary keys on demand when inter-node communication occurs. This design decision stands because a node can be the owner of various keys at the same time. Moreover, this

algorithm enables to parallelize the routing process in a tree-based way (performing a somehow similar process than in [50]). To do so, *CAPS only employ local routing information leveraging the underlying SPN*.

Algorithm 5.1 *subscribe*
/* Subscription installation. Parallelize the process given the subscription *subs*, installing it on rendezvous nodes defined by the corresponding keyset *KS*. */
Input: $n \leftarrow$ current node
Input: $KS \leftarrow$ selected keyset (for simplicity $[lowerBound\ ..\ higherBound]$)
Input: *subs* $\leftarrow$ subscription to install, including the subscriber ID

```
1:  localKS ← segment(n, n) ⋂ KS
2:  if localKS ≠ ∅ then /* n is a rendezvous node */
3:     store subs in n
4:  end if
5:  remainKS ← KS \ localKS
6:  neighbors ← getLinks(n)
7:  while remainKS ≠ ∅ do /* parallelize the subscription installation */
8:     neigh ← extract a node from neighbors
9:     neighKS ← segment(n, neigh) ⋂ remainKS
10:    if neighKS ≠ ∅ then
11:       subscribe(neigh, neighKS, subs)
12:       remainKS ← remainKS \ rnodeKS
13:    end if
14: end while
```

Algorithm *subscribe* (see Alg. 5.1) presents the parallelized *breadth-first*-like routing algorithm, employed in all subscription installations. Note that the function *segment(n, p)* shows the owned keyspace segment for node p, employing only local routing information of node n. The subscriber starts the subscription installation by calling to *subscribe(n, BM(subs), subs)*, where *subs* is the subscription and $BM(subs)$ produces the initial keyset *KS*. This call makes the node to send a set of *sub-keysets* (line 11) directly to node's neighbors, parallelizing the process from the very beginning. Upon reception of these messages, nodes employ the same algorithm to (i) store the given subscription iff they become rendezvous nodes (lines 1-4) and (ii) forwarding the subscription to the given sub-keyset (lines 7-14). There are no more forwardings when the sub-keysets are empty. As the sub-keysets become disjoint between them, this algorithm guarantees that the subscription installation is performed *visiting the nodes* involved in the operation *at most once*. In addition, as stated in Remark 2.2 at page 15, for any given key there will be one link in node's routing table that will be used to

forward to a given message. This means that for any remaining sub-keyset *remainKS* (line 5) there will be *at least* one link that would be used to forward to all keys in remainKS. This way, remainKS will be forwarded to through *at least* one link. This property ensures that the keyset is covered entirely and, thus, the algorithm always ends.

As aforementioned, this algorithm avoids building the whole keyset, which for high selectivity ratios would be unscalable. Instead, the processing node only builds the lower and higher bounds of the keyset: $[lowerBound\ ..\ higherBound]$. Note that these values are the result of applying the BM function to the lower and higher bounds per dimension of the subscriptions, respectively. Therefore, all keyset operations (i.e., difference '$\setminus$' and intersection '$\bigcap$') are actually translated to *low-cost keyspace segment operations*. In consequence, the information sent between nodes is reduced to the subscription, the subscriber ID and two keys for the segment definition.

The whole operation takes $O(\log N + \alpha)$ hops, with a maximum cost on message delivery dilation of $O(\log N)$, where N is the number of nodes within the network and α is the number of rendezvous nodes for the given subscription. Therefore, the inherent communication costs are reduced to the minimum expression and, consequently, this demonstrates the routing efficiency of our algorithm.

Subscription storage. Rendezvous nodes for any given subscription must store it locally for a later event matching. Depending on the context, note that nodes can store a notably amount of subscriptions. Thus, it is necessary to have an efficient local subscription index for, given an event, decide which is the set of matching subscriptions.

Unsubscription. In CAPS, subscriber is responsible of maintaining registered all Application current interests and unregister removed interests. Note that our system does not limit the number of subscriptions a subscriber maintains. Unsubscription process is realized in the same way than the subscription, but removing any locally stored subscription iff (i) the subscription (if present) matches the current unsubscription message, and (ii) the subscriber ID coincides.

5.2.2.2 Event Management

When Applications publish events, they are processed by the *Event management* block. Intuitively, given the Application's event, this block realizes the event mapping as seen in Section 3.3.1.1 and sends it to the rendezvous node. As CAPS maps an event into a single key, CAPS sends the event in a single message directly to the rendezvous

node, leveraging the underlying SPN routing infrastructure and, consequently, with a communication cost of $O(\log N)$ hops. The dissemination of the event through the corresponding subscribers is performed by the *Notification management* block.

5.2.2.3 Notification Management

The management of notifications involves two main operations: event matching and event notification to subscribers, which are detailed as follows.

Event matching. When a rendezvous node receives an event from a publisher, this node performs the event matching against the locally stored subscriptions. Some useful matching techniques appear summarized in [112], like decision trees or binary decision diagrams. For the sake of clarity, given a subscription selection $[0..14]$ in an attribute *price* with a domain of values $[0..3000]$, the given selection can also be defined as a filter $price < 15$. In consequence, decision trees or binary decision diagrams can be included in CAPS to perform efficiently the event matching. However, for simplicity of its implementation, we have used the brute force technique [112]. This technique tests the given event sequentially against all subscriptions. This has the advantage that this technique can be used with any kind of subscription. Regardless of the technique, the result of this operation is a list of subscriber IDs that are interested in the event and must be notified.

Given that both subscriptions and events are mapped into the SPN keyspace, it is not clear how the matching process could be accomplished in a distributed setting. Let us now put some light into that. Regardless of the mapping technique, whenever a subscription depicts interest in a given event, the mapping should guarantee that the given subscription is to be included into the matching process against such an event. As previously explained, the result of this process should be the list of subscriber IDs.

Let us now clarify the way events can be matched through suitable subscriptions. Let S and E be a subscription and an event, respectively, so that $E \in S$. This means that the subscription S covers the values specified in the event E (i.e., event E matches against subscription S). Let n be the rendezvous node for event E (i.e., the owner node of the mapped event key $k_E = BM(E)$), and let $ks_S = BM(S)$ be the mapped subscription keyset. Since $E \in S$, it is true that $k_E \in ks_S$, so that node n will store the subscription.

Upon reception of the event E, the node n will successfully match E against the locally stored subscriptions, particularly against S, and node n will include the subscriber of S into the list of subscriber IDs of nodes to be notified. For the sake of clarity,

let us propose a counterexample. Suppose that an event F is delivered to the owner node of its key $k_F = BM(F)$ and there is no subscription installed in that node. Clearly, this means that there is no subscriber interested in this kind of events. In this case, the costs of this event delivery is a negligible amount of $O(\log N)$ hops.

Algorithm 5.2 *notify*
/* Parallelized event notification. */
Input: $n \leftarrow$ current node
Input: $n.app \leftarrow$ current application using the pub/sub services of node n
Input: $ids \leftarrow$ list of subscribers interested in the event
Input: $event \leftarrow$ event to disseminate

```
1:  localIds ← segment(n, n) ⋂ ids
2:  if localIds ≠ ∅ then /* n is the owner of some ID(s) */
3:     if node.id ∈ localIds then /* n is a subscriber */
4:        notify event matching for n.app application
5:     end if
6:  end if
7:  remIds ← ids \ localIds
8:  neighbors ← getLinks(n)
9:  while remIds ≠ ∅ do /* parallelize event notification */
10:    neigh ← extract a node from neighbors
11:    neighIds ← segment(n, neigh) ⋂ remIds
12:    if neighIds ≠ ∅ then
13:       notify(neigh, neighIds, event)
14:       remIds ← remIds \ neighIds
15:    end if
16: end while
```

Event notification. CAPS performs the event notification process similarly to that of subscription installation. Specifically, Algorithm *notify* (see Alg. 5.2) realizes the event notification, starting the process from the event's rendezvous node. Instead of having a keyset KS, the algorithm uses the list of subscribers *ids* obtained from the step before. Besides, notifications will only reach those Applications (line 4) whose local node ID coincides with some ID from *ids* (line 3). Other IDs present in $localIds$ become, in fact, failed nodes that are not present in the SPN. Specifically, this design of the algorithm enables CAPS to recover from node failures (see Section 5.2.2.4). In summary, event notification inherits the same cost, i.e., $O(\log N + \alpha)$ hops, with a maximum cost on message delivery dilation of $O(\log N)$. As mentioned previously in the subscription

installation, the communication cost of our *SPN-generic algorithm* represents the minimum number of routing hops. The reason behind that is because there is no specific overlay to directly disseminate events, but we leverage an underlying peer-to-peer routing infrastructure, performing as efficiently as the SPN lets.

5.2.2.4 Failure Recovery

When nodes fail unexpectedly, CAPS must continue with all the system information consistent and up-to-date in order to operate correctly. As CAPS nodes can adopt the subscriber, publisher and rendezvous roles at the same time, we detail the tasks to be perform in every situation as follows.

If a *publisher fails*, it does not produce any effect into the system in terms of consistency. Clearly, the consequence is that new events will not be published, but the system itself remains consistent.

If a *subscriber fails*, the system contains with high probability (w.h.p.) some orphan subscriptions. It is easy to see that if we add extra functionality in Algorithm *notify* (see Alg. 5.2) between the lines 5 and 6, CAPS can send back a failure message to rendezvous nodes to mark all the subscribers present in the set $localIds \setminus \{node.id\}$ as failed. To do so, rendezvous nodes increment by one a per-subscriber failure counter. Later on, rendezvous nodes will remove all locally stored subscriptions from nodes with $\geq E$ failure marks. Moreover, this counter guarantees that subscriptions will remain in the system after little temporary disconnections. This mark counter is reseted when an event notification is performed successfully.

If a *rendezvous node fails*, CAPS would potentially lose the subscriptions the node stores, as well as future events will not be notified to all the expected subscribers. Given that CAPS leverages the SPN infrastructure for routing purposes, the system takes also advantage of SPN replication algorithms in order to place subscriptions copies in the corresponding nodes. For instance, replicas are usually placed in the following r successors in the SPN Chord [7]. This way, when a rendezvous node fails, all future events will arrive at the failed node's successor and, thus, no event notification is lost.

5.3 Evaluation

The theoretical analysis of the event (*object*) and subscription (*range object*) mapping was already addressed in Section 3.3. This analysis showed a good performance on

high-dimensional data domains. Let us now demonstrate its effectiveness by experimentation. To do so, the results of significant simulations will exhibit the good performance of CAPS in high-dimensional content-based event dissemination contexts, and in a wide variety of subscription selectivity ratios.

5.3.1 Experimental Setup

We implemented a prototype of CAPS, and we simulated it employing Chord [7] protocol as the SPN routing infrastructure. Chord nodes IDs are picked up uniformly distributed at random within the keyspace, and we built networks of up 1K, 5K and 10K nodes. For this prototype of CAPS, we built all node IDs of 28 bits (i.e., $m = 28$) in order to obtain time-efficient simulations.

Concerning the workload model, there is currently no publicly available data traces of real pub/sub applications [88]. Consequently, we tested CAPS with various synthetic datasets. Particularly, we characterize the sets of subscriptions with the following properties: *number of dimensions*, *selectivity ratio* and *subscription overloading*.

The number of dimensions vary from 2 to 14 and the corresponding number of mapping bits per dimension from 14 to 2 (i.e., *m/num. dimensions*). Recall that we do not simulate 1-bit mappings scenarios (i.e., 28 dimensions in this case), because of their poor precision. We specify a total selectivity ratio of 5%, 10%, 25% and 40%, taking the same ratio for every subscription's attribute. By subscription overloading we define the number of attributes from a subscription that must be mapped taking the higher bound of number of mapped values. We selected this number uniformly at random per subscription from $[1..D]$, where D is the number of dimensions. The rest of attributes are mapped having the lower bound on the number of mapped values.

We then characterize the sets of events with the following properties: *number of dimensions* and *matching ratio*. As before, the number of dimensions ranges from 2 to 14. Conversely, the matching ratio defines the probability of an event to match an arbitrary subscription. During the event set construction, we fixed this probability in such a way that (i) every event matches at least a subscription and (ii) every subscription is matched at least once.

As for the amount of events and subscription, we built 1K subscriptions for every number of dimensions and selectivity ratio (i.e., a total of 16 sets of 1K subscriptions each), and 1K events with the above properties for every subscription dataset. Even though these datasets could seem little, the performance evaluation of CAPS does not require bigger datasets. That is, bigger datasets would neither become more representative nor contribute in providing more insights of our approach. Subscribers

and publishers in every test are selected randomly from the network. For the sake of correctness, the results shown in this section correspond to an average of at least 10 executions.

We address the subscription and notification analysis through two main properties: *bandwidth scalability* and *memory scalability*. The *bandwidth scalability* evaluates the routing costs of the CAPS algorithms. To do so, we differ from *routing hops*, which are forwarding hops on *non*-rendezvous nodes, and *subscription* or *notification hops*, which are visited rendezvous nodes or visited subscribers receiving an event notification, respectively. We analyse in *memory scalability* the feasibility and load balancing of our system in terms of memory usage.

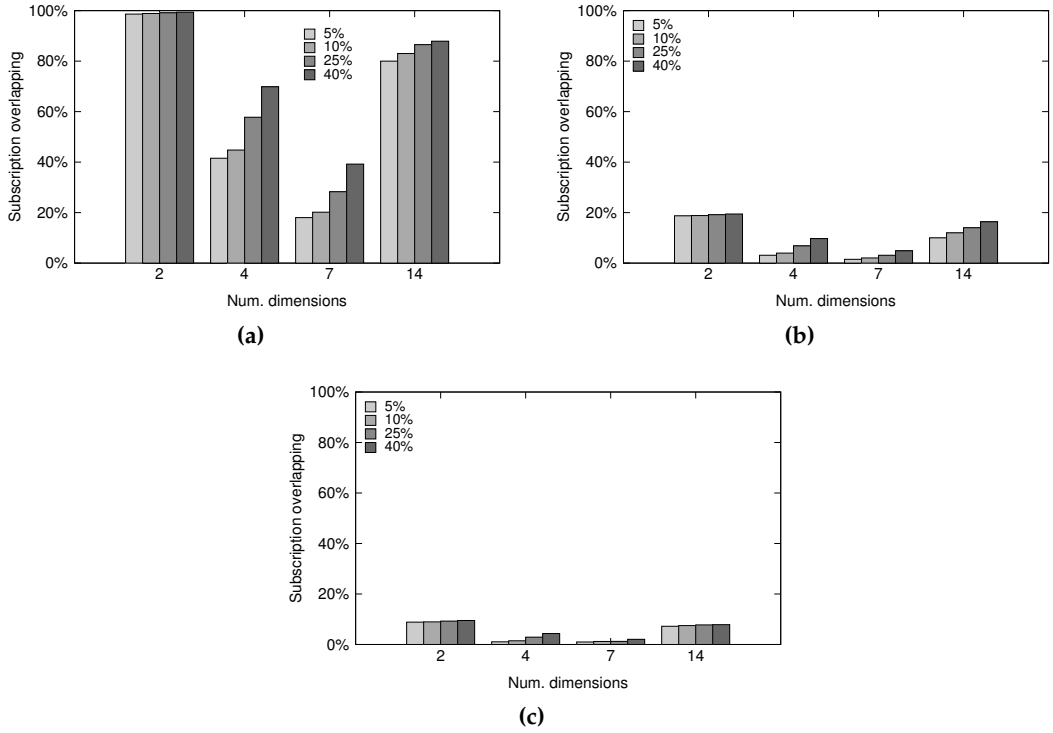


Figure 5.2: Ratio of rendezvous nodes per subscription. Network size: (a) 1K nodes; (b) 5K nodes; (c) 10K nodes.

5.3.2 Subscription Assessment

This section details the evaluation of CAPS regarding to the subscription installation process.

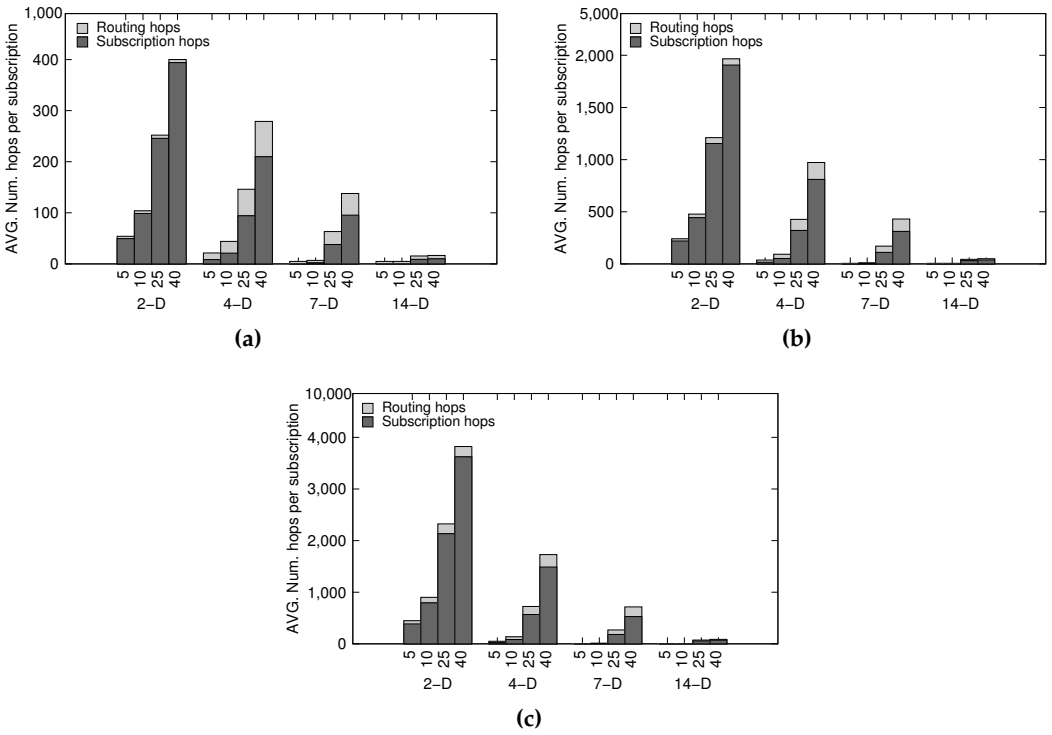


Figure 5.3: Average number of hops performed per subscription. Network size: (a) 1K nodes; (b) 5K nodes; (c) 10K nodes.

5.3.2.1 Bandwidth Scalability

As it is depicted in the Fig. 5.3, the Alg. 5.1 *subscribe reports almost no overhead* in terms of routing nodes, becoming a reduced ratio of the visited nodes. To evaluate more precisely this operation, we include in Fig. 5.2 the average (AVG) ratio of N_r/N_s, where N_r refers to the amount of rendezvous nodes per subscription and N_s sets the number of nodes laying between $[KS.lowerBound..KS.higherBound]$, i.e., laying between the keyset's minimum and maximum keys once the subscription is mapped. For instance, Fig. 5.2c depicts that more than 80% of nodes laying between the range $[KS.lowerBound..KS.higherBound]$ are not rendezvous nodes. Nevertheless, our subscription installation algorithm is not affected by this fact. Moreover, this results fits the estimated routing cost $O(\log N + \alpha)$ hops, where the routing nodes appear to be the factor $\log N$ and rendezvous nodes the term α.

In addition, the *high-dimensional context property* is also present in Fig. 5.3. In any network size, CAPS demonstrates the best performance on high-dimensional contexts, specially on the 14-D scenario (i.e., $b_i = 2$ bits) where very few nodes appear as rendezvous. For the same reason, under 2-D scenarios the selectivity ratio coincides with

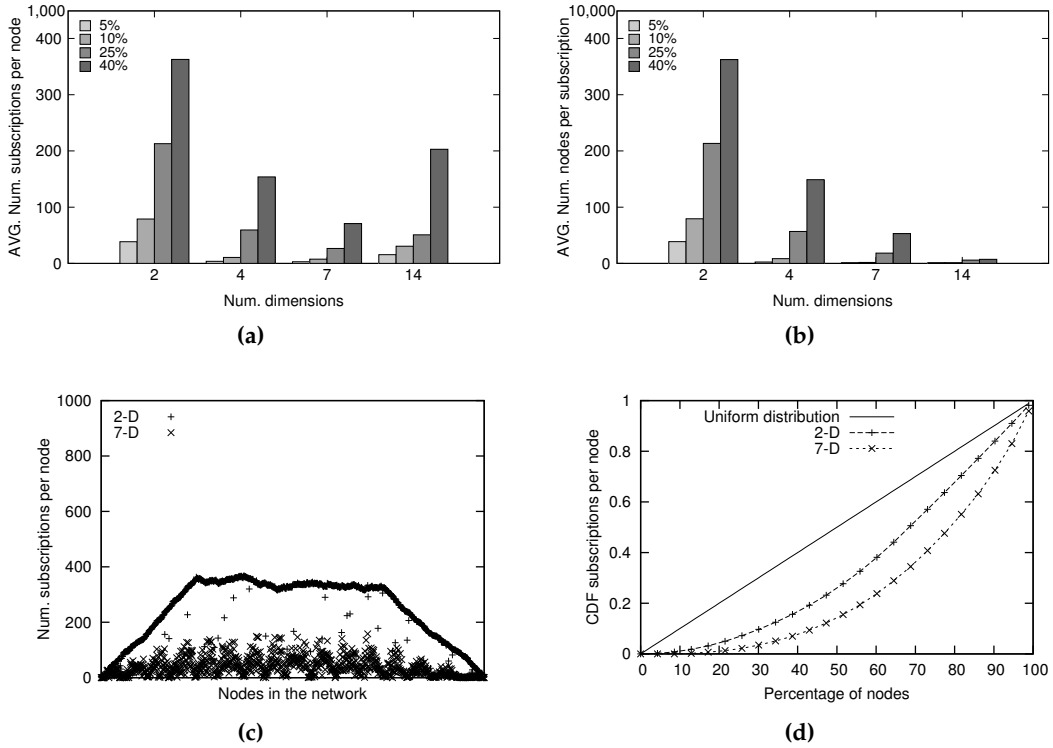


Figure 5.4: Subscription storage analysis in 10K-node networks. (a) Average number of stored subscriptions per node. (b) Average number of nodes a subscription is stored in. (c)(d) Distribution of subscription storage within the network (Selectivity ratio: 25%).

the ratio of rendezvous nodes within the network, defining CAPS as a non proper solution for low-dimensional scenarios ($D < 4$).

5.3.2.2 Memory Scalability

We analyse now the memory usage by subscriptions once all they have been installed, focusing on *the number of subscriptions* a node stores, *load balancing* through the network, and *the number of nodes* a subscription is installed in. As long as the behavior shown in all network sizes were very similar, we only include the results for the 10K-node network.

Fig. 5.4a depicts the average number of installed subscriptions per node. As expected, the greater dimensionality of subscriptions, the less amount of subscriptions are stored at rendezvous nodes. The slight increase in the amount of subscriptions for 14 dimensions appears because of the dataset instead of the algorithm. Specifically, the randomly built subscription datasets for 14 dimensions experience a great overlapping ratio (see Fig. 5.2). For instance, for 1K nodes and 14 dimensions (Fig. 5.2a),

more than 80% of nodes are rendezvous, regardless of the selectivity ratio. This will be clearly translated to more nodes storing the same subscriptions. However, we preferred do not change these datasets in order to present more realistic results.

Complementary to the above analysis, Fig. 5.4c and Fig. 5.4d present the distribution of subscriptions through the network. In Fig. 5.4c there exists a clear difference on the behavior of the 2-D scenario (upper curve) and the 7-D scenario (lower zigzag). While almost all nodes in the 2-D scenario are storing the same considerable amount of subscriptions in a *continuous* way, the 7-D scenario presents a *discontinuous* storage which enables CAPS to load the balancing through all nodes. Moreover, if SPN replication mechanisms are applied into CAPS, immediate neighbours of the corresponding SPN nodes appear to be memory-available and, thus, replication does not overload them. Fig. 5.4d depicts the cumulative distribution function (CDF) on the amount of installed subscriptions, having the theoretical *Uniform distribution* as the upper bound on both cases. For instance, the 20% of the nodes (i.e., 2K nodes) in a 7-D scenario are rendezvous nodes for almost the 45% of the subscriptions. This fact enables CAPS to support a great amount of subscriptions within the system, without overloading the whole system.

The number of keys a subscription is mapped into, greatly concerns the system load induced by a single subscription. As expected, the greater the subscription dimensionality, the smaller number of keys a subscription is mapped into. Moreover, since nodes are uniformly distributed at random along the SPN keyspace, when the mapping produces less number of keys, the number of nodes that become rendezvous is also reduced, as Fig. 5.4b depicts. CAPS benefits also from high-dimensional contexts so as to produce small global subscription load and, therefore, to support a high number of subscriptions.

5.3.3 Notification Assessment

As long as event delivery takes a single SPN message routing in order to reach the rendezvous node, this is a straightforward and lightweight operation and we include this cost as part of the analysis of the overall notification process (i.e., from rendezvous nodes to subscribers), which is presented in the following.

5.3.3.1 Bandwidth Scalability

Fig. 5.5 depicts the close behavior of the Alg. 5.2 *notify* in all scenarios. Firstly, we can observe the *completeness* of the algorithm, given that the same amount of subscribers

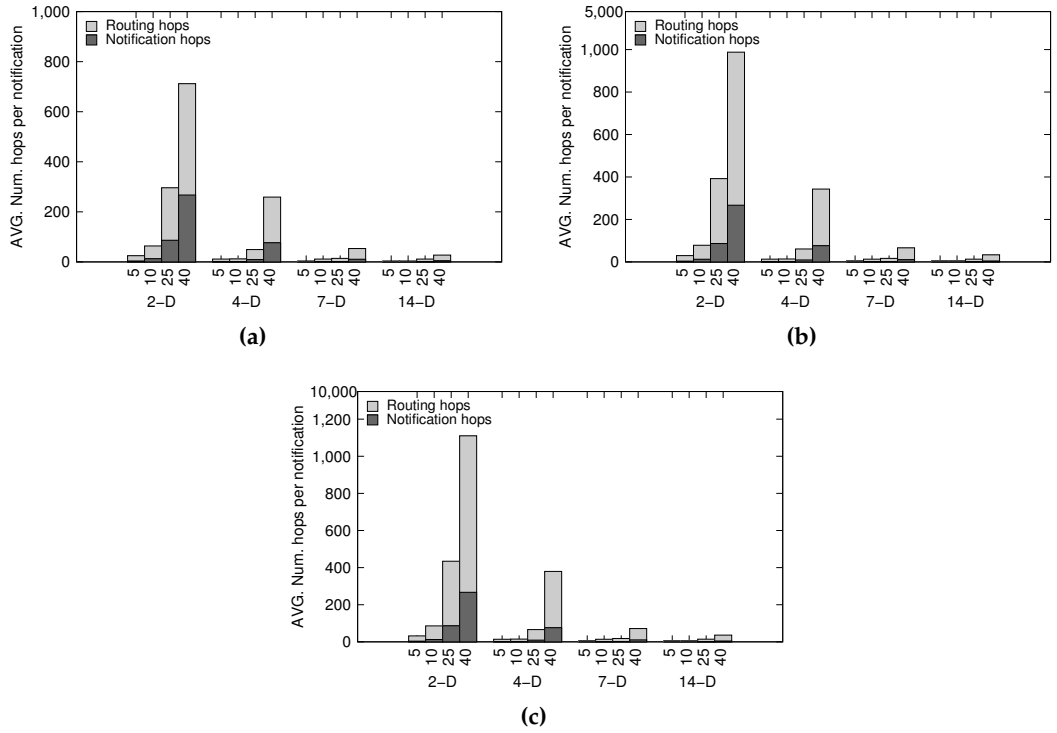


Figure 5.5: Average number of hops performed per notification. Network size: (a) 1K nodes; (b) 5K nodes; (c) 10K nodes.

are notified (i.e., *notification hops*) in all scenarios, but varying only the network size. Remember that for all scenarios there exist 1K subscriptions performed by different nodes. Secondly, the total number of hops increases logarithmically when the network size increases. To understand why, we have to consider that IDs from subscribers are not correlated, in clear contrast with what occurred with subscriptions. Given that CAPS leverages the SPN routing infrastructure, there is no way to improve these results. Nevertheless, as long as links on SPN nodes can be selected in terms of proximity (e.g., latency [8]), the response time experienced by this routing cost can be greatly mitigated by selecting the SPN properly and its configuration.

5.3.3.2 Memory Scalability

We analyse now the memory usage by notifications, focusing on the CAPS load in terms of *number of event matchings* per rendezvous node and *load balancing*. From Fig. 5.6 we can see that, while network size increases and, then, more nodes share the rendezvous responsibility for any given segment of the keyspace, the average number of events arrived at rendezvous nodes decreases notoriously, providing an implicit *load balancing* mechanism behind the CAPS design.

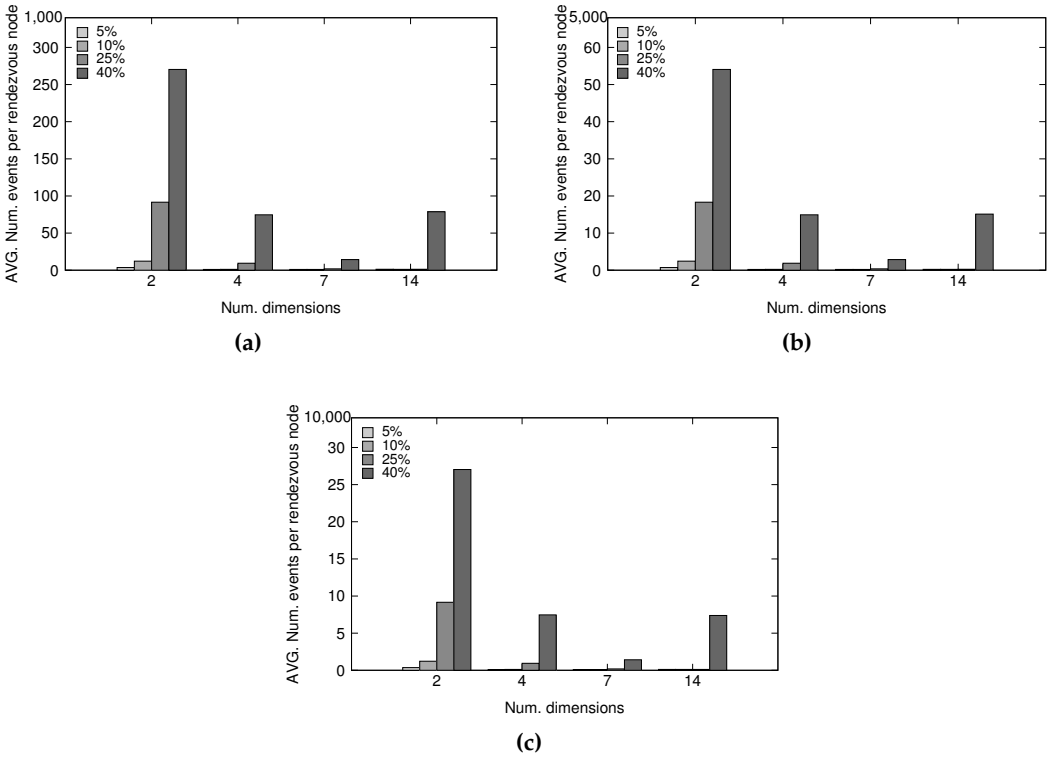


Figure 5.6: Average number of events that reaches rendezvous nodes. Network size: (a) 1K nodes; (b) 5K nodes; (c) 10K nodes.

Complementarily, Fig. 5.7 depicts, for a 10K-node network and 25% of selectivity ratio, the event reception at rendezvous nodes throughout the system. We omit other network sizes because the results were very similar. It can be seen from Fig. 5.7a than for the 2-D scenario, almost all nodes have between 40 and 120 event receptions, while in the 7-D scenario almost every node has less than 5 receptions. Given that rendezvous nodes have also less stored subscriptions in high-dimensional contexts, CAPS gains event matching speedup and reduction on memory consumption. In addition, as seen in Fig. 5.7b, the system experiences a high degree of *event reception fairness* (i.e., so that the amount of event receptions at rendezvous nodes is shared equitably).

5.4 Conclusions

In this work we have presented CAPS, the design and evaluation of a service for multi-dimensional content-based event dissemination pub/sub systems for our framework, which, therefore, validates our sixth contribution of our thesis. This work was outlined in [113] and then the whole work was introduced in [111].

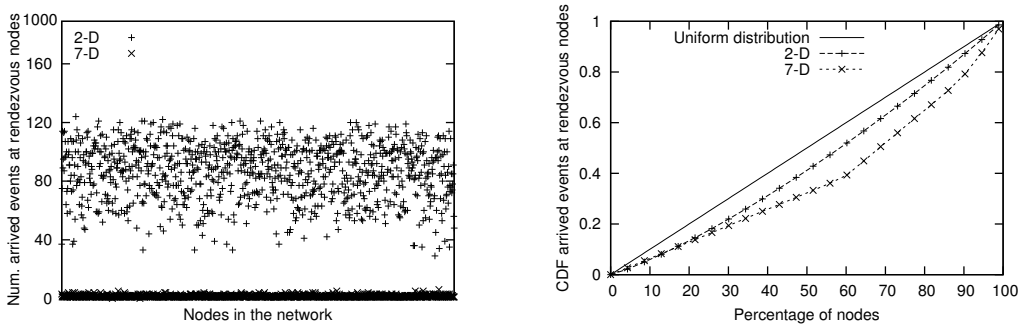


Figure 5.7: Distribution of event reception at rendezvous nodes within the network (Selectivity ratio: 25%). Network size: (a) 10K nodes; (b) 10K nodes CDF.

As it was outlined, the key pub/sub service design relies on the *data adaptation module*, and with the algorithms (based on the data adaptation technique) to set up subscriptions and disseminate new events. In particular, our framework and CAPS are *lightweight* because our approach does not need to build specific pub/sub overlays onto SPNs. Instead, our system employs the generic algorithms presented in the data adaptation module. They are actually based on the *rendezvous model* and leverage the underlying SPN routing infrastructure. This guarantees that the service, as well as our framework, is *SPN-generic and portable*.

Additionally, the CAPS evaluation has been addressed through a *formal analysis of the performance and* necessary *trade-offs*, for a correct performance of this rendezvous-based approach. Both theoretical and experimental results demonstrate that our system is *efficient in terms of communication cost and with low memory usage*, selecting very few nodes as rendezvous nodes. The results also show that CAPS provides its best performance on low selectivity ratios for any dimensionality and especially on *high-dimensional contexts*, thanks to the BM function of our data adaptation module.

Above all, this work demonstrates that deploying odd, complex, high-level services into our framework is *feasible*. In the present case, CAPS has shown extensively its good properties, such as *efficiency*, *scalability to big peer-to-peer networks* and to *high-dimensional data domains*, as well as a good *data and routing load balancing* throughout the network. One aspect does not treated in this work, though, is the effect in our system of nodes joining and leaving the network. This will be one point to address in our future work.

6

Conclusions and future work

Finally in this thesis, we present some concluding remarks and the results of this thesis in the following section. In addition, we detail some future research work in the field of this thesis in Section 6.2.

6.1 Conclusions and outcomes

This thesis has been motivated by the lack of genericity on the way services, such as data management services (e.g., range queries or spatial queries), or content distribution services (e.g., publish/subscribe services), are currently designed and deployed into structured peer-to-peer networks (SPNs). To properly evaluate how, we have presented **a complete analysis on the state of the art in both data management and content distribution peer-to-peer-based services** (see Chapter 2). We have constructed an evaluation framework with up to 10 properties in order to build an objective and fair comparison among the presented systems. In addition, this analysis has provided the necessary insights and has brought to light the *shortages* of the existing solutions: *maintenance overhead*, *lack of generic solutions*, *non portable services and applications*, and *lack of structural support for multiple services* at the same time. Let us now explain what these issues imply for a system.

- *Maintenance overhead*. This is one of the first shortages we detected in existing solutions. Building new SPNs for new services (usually over already existing P2P substrates) is a common practice. However, when nodes join or leave, or when nodes and links fail, all the overlays experience an important cost on signaling traffic due to overlay maintenance (such as fixing the routing table or looking for new neighbors to replace the failed ones), which (on the whole) leans to *duplicated costs*.

This motivated us to look for **an alternative**. Some other works favored the reutilization of existing SPNs, by adapting the application data domain to the keyspace of the SPN. Clearly, this kinds of solutions introduced a certain **genericity**. That is, we

could see that several applications could be deployed onto a single SPN. However, all these applications and services are designed to work specifically over the given SPN, so that the *genericity* is partially *truncated*. We believe that **applications should be easily deployed over most of the SPNs** without much modification or knowledge of the underlying SPN, while keeping the efficiency and scalability of the solution.

To reach such a genericity, two other shortages come to light:

- *Lack of application and service portability*. In other words, existing solutions do not guarantee a full genericity and portability of applications among SPNs.
- *Non coexistence of multiple services*. Complementary to the above issue, the approach we are looking for, should promise that different applications and services could be deployed over the same SPN instance.

We have addressed this lack of genericity and all the aforementioned shortages by defining a framework which (i) can be deployed in most of the SPNs (i.e., **SPN-generic**), and (ii) provides a **set of tools to design and deploy new services**, so that these services are thereafter offered to any end-user application. That is, since services are deployed into our framework, which is fully portable among SPNs, these services become portable and can be utilized by most of the SPNs.

To do so, we firstly determined the common set of properties of the targeted SPNs in this thesis (see Section 2.1.1). With these details kept in mind, as well as with the analysis of the data management and content distribution services (see Chapter 2), we then defined the whole **framework structure** at Chapter 3.

In particular, to facilitate the distributed communication of our framework, it only **requires a minimum set of functionality** (easily or already) provided by SPNs, which is composed by *sendMessage*, *getLinks* and *deliverMessage* functions. From an architectural point of view, **the framework is composed by several modules** (see Fig. 6.1), each of which is responsible for addressing specific challenges. Let us introduce them in the following lines.

Data adaptation module. This is the key component of our framework. It is worth noting that the considered SPNs have one-dimensional keyspaces, so that the *multi-dimensional data domains of the applications are not directly supported*. This module addresses the transformation of the multi-dimensional data domain to a unique representation. The representation domain coincides with the keyspace of the SPN, so that it can be processed transparently and elegantly by the SPN.

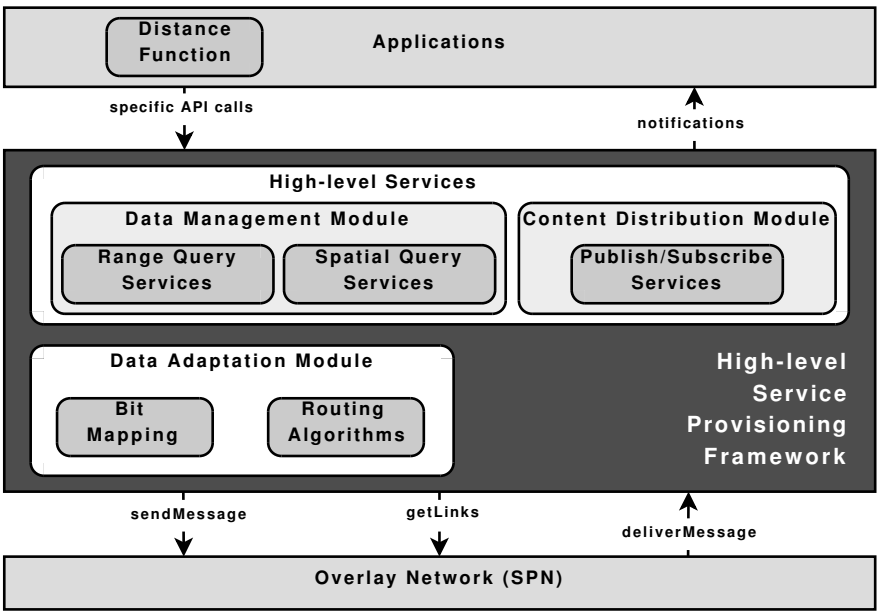


Figure 6.1: Structure of our framework provisioning high-level services. All the modules and information flow are detailed.

To deal with this data transformation, **we constructed a dimensionality reduction technique**, namely **Bit Mapping (BM)** –see Section 3.3. Note that BM is generic to any kind of application data domain, motivated by the fact that our framework must support multiple services and applications at the same time. In particular, the BM design and development was due to that the existing hash functions do not fulfill all the requirements expected by our approach, such as *data and routing load balancing*, while range-based operations are *efficient*ly supported.

In addition, we outlined how distributed operations (e.g., insertions or complex range-based operations) are to be addressed by the services settled down in our framework. In other words, **we designed the necessary algorithms to support complex operations** into our framework. To do so, **we do not build new overlays**, but leverage the underlying SPN and its routing capabilities to complete such operations.

These algorithms are then formally evaluated through a theoretical analysis, as well as by an experimental assessment (in the use cases presented by the other modules). **Our routing algorithms** demonstrated to **perform efficiently** in a wide variety of scenarios, **and** to **scale** in the number of nodes, as well as in the number of dimensions of the application data domain.

Data management module. This module presents **two use cases** of service deployment into our framework. The first is the design, development and evaluation of a service that provides **multi-dimensional range queries**. Multi-dimensional data is stored into the SPN. To do so, we use the BM function to produce the related keys. The nodes responsible for these keys store the data.

We then evaluated the performance of our multi-dimensional range query service through representative simulations, demonstrating the efficiency not only of the BM adaptation function, but also of the algorithms described in the data adaptation module. They together depict an excellent **data and routing load balancing** throughout the network of nodes, compared to existing solutions.

A **geographical information service** is the second use case presented in this module. We categorize the problem and all the challenges to be addressed, such as *multi-dimensional data* or *data location*. To successfully address this geographical location service, we introduce a novel approach where **both nodes and data are organized into clusters** of nearby items. To do so, we slightly alter the BM adaptation function to encode the geographical information.

In addition, we also present a particular SPN hierarchical structure, which benefits to the clusterization of nodes and data. Data is indexed in the same way as before; that is, data is stored in the nodes responsible for the produced keys after the data adaptation. **Exact match queries**, **spatial queries** and a novel **geocast query** are introduced. Even though the application data domain is multi-dimensional (location and semantic description), our approach is able to answer to exact match queries very efficiently. Actually, an exact match can be seen as a *get* operation in a DHT. Spatial queries are deeply analysed, demonstrating the efficiency of our approach. Instead, the geocast queries retrieve the same kind of information from several locations at the same time with a **very low cost**, compared to other systems. To conclude with, the clusterization of nodes and information also benefits providing **path locality**, as well as **content locality**, which lean to boost the **scalability** of our approach. The reader can find all these results in Sections 4.1.4 and 4.2.6, respectively.

Content distribution module. Not only is our framework capable of managing data in a distributed setting, but also it can distribute content. In particular, the use case addressed in this module is a **multi-dimensional content-based publish/subscribe service**. Employing the same idea as before, **we adapted events and subscriptions to the SPN keyspace** by means of the BM function. Consequently, we demonstrated that

if an event E matches a subscription S, the corresponding E's transformation key will appear in the set of keys from the S's adaptation keys.

The subscription management is addressed as follows. Since we leverage the routing capabilities of the underlying SPN, we employ the **rendezvous model** to meet events with subscriptions in the distributed setting. In particular, subscriptions are stored in all nodes responsible for their adaptation keys, employing the same routing algorithms from our data adaptation module. Thereafter, events are merely routed to their rendezvous nodes. These nodes will match the incoming events with the locally stored subscriptions and, afterwards, they will start the notification phase. In this phase, employing a slight modification of the routing algorithms, all interested subscribers will receive the new events.

The whole set of operations are always performed with only the **local knowledge** of the nodes, with the adaptation technique and routing algorithms presented in the data adaptation module. By experimentation conduced by simulations, we demonstrate that our approach successfully address this challenge. Indeed, the service depicted good **data and routing load balancing** properties, as well as a **scalability** in terms of number of nodes and the number of dimensions of the application data domain, as described and extensively shown in Section 5.3.

Concluding remarks. To sum up with this thesis, we detail the set of good qualities of our framework and the services deployed upon it:

- **Generic.** We provide two ways of genericity. Several services can be deployed into our framework. Since our framework is portable among SPNs, the services become SPN-generic. Conversely, by employing a minimum set of functions, our framework can be deployed onto most of the SPNs.

- **Portable.** As a consequence of the above point, our framework, as well as all services and the end-user applications become portable among SPNs. This should facilitate building largest communities of users collaborating in the system, so that more data and process resources will be made available.

- **Lightweight.** All the distributed functions employ only local state information of nodes (i.e., no global data structure is required), what leans to a lightweight approach. Indeed, since we do not construct further overlays to implement our services, we remove duplicated complexities and costs related to the overlay maintenance, favoring again a lightweight solution.

- **Support of different high-level services.** At the same time, we have demonstrated that our framework can support several services. At the same time, these services provide their functionality to end-user applications.
- **Complex data domain support.** We have presented three use cases with odd application data domains. They had in common that all they were multi-dimensional, so that any data object is formed by a list of terms or attributes. Conversely, they differed in the data type that the objects consists of (e.g., numerical or string).
- **Efficient data adaptation.** The data transformation from a multi-dimensional domain to a uni-dimensional one could present several issues, such as poor balancing of data storage among nodes. However, all the services presented in this thesis demonstrate an excellent data load balancing throughout the network, improving the results of other existing systems. It is worth noting that we did not use data load balancing techniques, such as caching or replication.
- **The higher dimensionality, the better.** In all the studied scenarios, we have compared and evaluated different dimensionalities. We have demonstrated that our data adaptation technique performs better (i.e., it has lower overheads) when the dimensionality is greater. This comes in strong contrast with other existing systems, which their performance decreases as long as the number of dimensions increases.
- **Complex operation support.** We have addressed complex operations into our services, mainly range-based operations. To do so, we have designed some SPN-generic routing algorithms that successfully deal with such kind of operations. Since they are generic of the SPN, we did not evaluate the absolute number of visited nodes, but the ratio of routing nodes (i.e., the nodes that are visited merely as operation forwarders), with respect to the number of targeted nodes. We have demonstrated that our generic algorithms perform well, inducing a very low overhead in terms of routing nodes, as well as good routing load balancing.

6.2 Future research lines

The work of this thesis has started a promising path through a high-level service standardization for peer-to-peer networks, unique in its design to the best of our knowledge. Actually, there are lots of systems deploying such kinds of services onto different SPNs. However, there is not a clear will in the research community to make

standardizations to facilitate an extensive use of the overlay networks. Only very few examples appears in our every-day experience, such as BitTorrent.

Initially, there were some attempts of building SPN-generic services, such as PHT (which built a trie over a DHT), SkipIndex (which supported several services onto it) or the Common API (which was addressed on providing the set of functions that any key based routing peer-to-peer infrastructure should provide to applications). However, none of the existing approaches provided the degree of genericity illustrated in our thesis, as well as the rest of properties on the whole. For instance, PHT had a high cost when iterating over consecutive values, whilst SkipIndex supported this operation efficiently for several services, but only tied to SkipIndex itself. The Common API, though, makes applications to know how the overlay works to efficiently develop services onto it. In addition, it rapidly became unused and there exist only few testimonials (such as FreePastry or PlanetSim).

We envision a stronger effort from the community for building standards for distributed services and applications (maybe *de facto*), that will help to all partners in the play. For instance, end-user applications could have more stable services with a reduced API, common to all services of the same kind (e.g., range queries or publish/subscribe services). However, in clear difference from the services developed up to now, designers of such applications would have to consider only the functionality they get from the services, but none of the insights from any different underlying peer-to-peer network. As another example, let us consider the actual tendency where any new distributed application is deployed with its own overlay infrastructure (i.e., one overlay per service). In the standardized scenario, end users would benefit from the fact that they would not need up to tens of overlays, since a single overlay would support several applications at the same time.

Other strengths come to light too. Since the number of overlays would be reduced, the users would become more stable, because the same overlay would be utilized for several purposes. Therefore, overlays would become as an almost permanent distributed communication infrastructure. In consequence, the communities of users would be larger, where more resources of any kind (e.g., files, CPU cycles, memory) and information would be shared among users. In addition, network providers would benefit because the bandwidth due to the overlay maintenance and specific overlay signaling traffic would be drastically reduced.

As a more close challenges, though, we detail in the following lines some of the issues to be addressed, in order to construct a further generic and usable framework. We classify them into two big sets: *research* and *development* tasks. The former has a

proper scientific value so as to address its research. Conversely, the latter would help to add new functionalities to our framework.

Research lines:

- **New algorithms.** The presented distributed algorithms deal with range-based operations. To prove them, we have evaluated them into three different use cases. However, other kind of services (such as information aggregation functions or k-nearest neighbors) will require new algorithms to successfully address them.

  In particular, information aggregation services (like MIN, MAX, AVG or VAR) necessitate a somehow global knowledge of the existing data distributed among nodes. But, there are alternatives to calculate them efficiently in a parallel, distributed way. Even though that the algorithms will have to be SPN-generic, their performance will vary depending on the underlying SPN. For instance, our hierarchical SPNs presented in Chapter 4 could greatly help to reduce the SPN communication effort when designing the algorithms to implement such functions.

  The key idea behind these new services is to develop new communication mechanisms that, based on the local state of nodes, will construct a consecutively more global knowledge system. Gossip-inspired algorithms could successfully address (at least partially) these new challenges. However, the data load balancing and data locality presented by our data adaptation technique, will boost its design and development, as well as the efficiency of such new algorithms.

- **Validation in real systems.** The use cases presented in our thesis are evaluated through simulation results of synthetic datasets with different distributions. Even though the distributions employed (like Zipf) correspond to real-life event distributions, they are not evaluated with real-life information.

  We believe that an extensive validation through (i) deployment in real testbeds (such as Planetlab), and (ii) simulation using traces of existing applications, will probably provide more insights of our solution, and will strongly demonstrate the feasibility and efficiency of our data adaptation technique, as well as of all the shown algorithms.

Development focus:

- **High-level abstractions.** We have demonstrated it is feasible that several services can be deployed into our framework. However, any different service provides its own particular API to end-user applications. In addition, different implementations of the same kind of service can offer (slightly) different APIs. In consequence, this motivates that there is almost no guarantee of the portability of the end-user applications among services. In the same line that was designed our framework, we foresee that a minimum set of functions could provide the whole set of operations, in an extensible and flexible way.

  To address this issue, some parameters will be necessary to consider, such as the durability of the information, the kind of information, or the model of information retrieval (e.g., *pull* for traditional queries, or *push* for publish/subscribe services). After this analysis, two basic functions would be clearly identified: *insertion* and *retrieval*, which will take all the detected parameters. The idea behind these parameters is that different configurations will behave as, for example, an exact match query or a range query. Moreover, a certain parameter setting for a *retrieval* operation in a *pull* mode would work as a range query; however, the same setting in a *push* mode would behave as a publish/subscribe system, so that nodes would be notified by the system when new data was available.

  We believe that this further abstraction for end-user applications will be very beneficial. Applications will be designed assuming the same kind of abstractions, what will ensure their genericity and portability among lots of architectures.

- **Network churn.** From the analysed systems, we have motivated the use of the underlying SPN to support our framework, instead of building new overlays for any new service. However, this thesis did not consider the effect of nodes joining and leaving the network. Thus, properties such as data consistency or durability are one of the following issues to address within our framework.

  The services presented in this thesis are based in the same family of algorithms. The same thing will happen for other new services. Therefore, the set of techniques to guarantee (for instance) data availability could be common for several services. Caching and replication techniques are traditionally the way that peer-to-peer data networks secure data availability, durability and consistency. In consequence, according to our data adaptation mechanism, new caching and replication services could be deployed into our framework.

Thanks to these services, for instance, end-user applications could specify a degree of data availability when interacting with our framework. In other words, applications could tell whether the information being stored is critical (where data should become durable and consistent with very high probability) or is not essential (i.e., stateful only at a certain degree).

We have demonstrated that our framework is feasible. However, the addition of caching and replication services into our framework, would make our approach more practical and deployable into a real-life system.

References

[1] Flikr. http://www.flickr.com/, 2009.

[2] Flikr: Maps. http://www.flickr.com/tour/maps/, 2009.

[3] Flikr: Keep in touch. http://www.flickr.com/tour/keepintouch/, 2009.

[4] Open photo community: Breaking beyond flickr. http://www.hockleyphoto.com/open-photo-community-distributed-flickr/, 2009.

[5] Napster. http://free.napster.com/, 2009.

[6] Gnutella. http://en.wikipedia.org/wiki/Gnutella, 2009.

[7] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pages 149–160, 2001.

[8] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM International Conference on Distributed Systems Platforms*, volume 2218, pages 329–350, November 2001.

[9] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Berkeley, CA, USA, April 2001.

[10] Bamboo Distributed Hash Table. http://bamboo-dht.org/, 2009.

[11] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[12] M. Frans Kaashoek and David R. Karger. *Peer-to-peer Systems II*, volume 2735/2003 of *Lecture Notes in Computer Science*, chapter Koorde: A Simple Degree-Optimal Distributed Hash Table, pages 98–107. Springer Berlin / Heidelberg, 20-21 February 2003.

[13] G. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, pages 10–10, Seattle, WA, USA, March 2003.

[14] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: A self-organizing structured p2p system. *SIGMOD Record*, 32(3):29–33, September 2003.

[15] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on xor metric. In *Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, March 2002.

[16] Venugopalan Ramasubramanian and Emin Gün Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI '04)*, pages 8–8, Berkeley, CA, USA, 29-31 March 2004. USENIX Association.

[17] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with coral. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI'04)*, pages 18–18, Berkeley, CA, USA, 2004. USENIX Association.

[18] Emil Sit, Andreas Haeberlen, Frank Dabek, Byung-Gon Chun, Hakim Weatherspoon, Robert Morris, M. Frans Kaashoek, and John Kubiatowicz. Proactive replication for data durability. In *5th International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, February 27-28 2006.

[19] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam. psearch: Information retrieval in structured overlays. *ACM SIGCOMM Computer Communication Review*, 33(1):89–94, 2003.

[20] Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. Globase.kom - a p2p overlay for fully retrievable location-based search. In *Proceedings of the Seventh IEEE International Conference on Peer-to-Peer Computing (P2P '07)*, pages 87–96, Washington, DC, USA, September 2-5 2007. IEEE Computer Society.

[21] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489–1499, 2002.

[22] E. Anceaume, M. Gradinariu, A. K. Datta, G. Simon, and A. Virgillito. A semantic overlay for self-* peer-to-peer publish/subscribe. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (IDCDS'06)*, page 22, 2006.

[23] Zhichen Xu and Zheng Zhang. Building low-maintenance expressways for p2p systems. Technical Report HPL-2002-41, HP, 2002.

[24] Fidel Cacheda, Vassilis Plachouras, and Iadh Ounis. A case study of distributed information retrieval architectures to index one terabyte of text. *Information Processing and Management: An International Journal*, 41(5):1141–1161, 2005.

[25] Odej Kao. A prototype for a distributed image retrieval system. In *Proceedings of the 9th International Conference on High-Performance Computing and Networking (HPCN Europe '01)*, pages 579–582, London, UK, 2001. Springer-Verlag.

[26] Nicholas Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, pages 113–126, March 26-28 2003.

[27] Farnoush Banaei-Kashani and Cyrus Shahabi. Swam: A family of access methods for similarity-search in peer-to-peer data networks. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM' 04)*, pages 304–313, New York, NY, USA, November 08-13 2004. ACM.

[28] Chi Zhang, Arvind Krishnamurthy, and Randolph Y. Wang. Skipindex: Towards a scalable peer-to-peer index service for high dimensional data. Technical Report TR-703-04, Princeton University, May 2004.

[29] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: supporting scalable multi-attribute range queries. *ACM SIGCOMM Computer Communications Review*, 34(4):353–366, 2004.

[30] M. Cai, M. Frank, J. Chen, and P. Szekely. Maan: a multi-attribute addressable network for grid information services. In *Proceedings of Fourth International Workshop on Grid Computing*, pages 184–191, November 2003.

[31] Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl

Aberer. Range queries in trie-structured overlays. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 57–66, Washington, DC, USA, August 31 - September 2 2005. IEEE Computer Society.

[32] Yanfeng Shu, Beng C. Ooi, Kian-Lee Tan, and Aoying Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Proceedings of Fifth IEEE International Conference on Peer-to-Peer Computing (P2P '05)*, pages 173–180, Washington, DC, USA, 2005. IEEE Computer Society.

[33] H. Sagan. Space-filling curves. In *Springer-Verlag*, 1994.

[34] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems (PODS '84)*, pages 181–190, New York, NY, USA, 1984. ACM Press.

[35] D. Hilbert. Ueber stetige abbildung einer linie auf ein flashenstuck. mathematishe annalen. *Mathematishe Annalen*, pages 459–460, 1891.

[36] Christian Bohm, Gerald Klump, and Hans-Peter Kriegel. Xz-ordering: A space-filling curve for objects with spatial extension. In *SSD'99: Proceedings of the 6th International Symposium on Advances in Spatial Databases*, volume 1651 of LNCS, pages 75–90. Springer-Verlag, July 1999.

[37] F. Dabek, B. Y. Zhao, P. Druschel, J. Kubiatowicz, and Stoica I. . Towards a common api for structured peer-to -peer overlays. In *Proc. IPTPS'03*, February 2003.

[38] Freepastry. `http://freepastry.org/FreePastry/`, 2009.

[39] Jordi Pujol-Ahulló, Pedro García-López, Marc Sànchez-Artigas, and Marcel Arrufat-Arias. An extensible simulation tool for overlay networks and services. In *Proceedings of 24th Annual ACM Symposium on Applied Computing (SAC' 09)*, pages 2072–2076, New York, NY, USA, March 8-12 2009. ACM.

[40] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical Report Technical Report, IBM Ltd., Ottawa, Canada, 1966.

[41] Edward A. Fox, Qi Fan Chen, Amjad M. Daoud, and Lenwood S. Heath. Order-preserving minimal perfect hash functions and information retrieval. *ACM Transactions on Information Systems (TOIS)*, 9(3):281–308, 1991.

[42] Marc Sànchez Artigas, Pedro García López, Jordi Pujol Ahulló, and Antonio F. Gómez Skarkemta. Cyclone: a Novel Design Schema for Hierarchical DHTs. In *Proceedings of The Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 49–56, August-September 2005.

[43] Marc Sànchez-Artigas. *A Hierarchical Framework for Peer-to-Peer Systems: Design and Optimizations*. PhD thesis, Universitat Pompeu Fabra, January 2009.

[44] Marc Sànchez Artigas, Pedro López García, and Antonio F. G'omez Skarmeta. A comparative study of hierarchical dht systems. In *Proceedings of the 32nd IEEE Conference on Local Computer Networks, 2007 (LCN '07)*, pages 325–333, 15-18 October 2007.

[45] Luis Garces-Erice, Ernst W. Biersack, Keith W. Ross, Pascal A. Felber, and Guillaume Urvoy-Keller. Hierarchical p2p systems. In Harald Kosch, Laszlo Boszormenyi, and Hermann Hellwagner, editors, *Proceedings of ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par)*, volume 2790 of *LNCS*, pages 1230–1239, Klagenfurt, Austria, August 2003. Verlang.

[46] D. A. Tran, K. A. Hua, and T. T. Do. A peer-to-peer architecture for media streaming. *IEEE Journal on Selected Areas in Communications*, 22(1):121–133, Jan. 2004.

[47] Yincui Hu, Yong Xue, Jianqin Wang, Xiaosong Sun, Guoyin Cai, Jiakui Tang, Ying Luo, Shaobo Zhong, Yanguang Wang, and Aijun Zhang. Feasibility study of geospatial analysis using grid computing. In *4th International Conference on Computational Science (ICCS' 04)*, pages 956–963, June 6-9 2004.

[48] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS)*, 30(2):364–397, 2005.

[49] David Novak and Pavel Zezula. M-chord: A scalable distributed similarity search structure. In *Proceedings of the 1st International Conference on Scalable Information Systems (InfoScale '06)*, page 19, New York, NY, USA, May 30 - June 01 2006. ACM.

[50] S. El-Ansary, L. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks. In *Proceedings of Second International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.

[51] Vittoria Gianuzzi, Alessio Merlo, Andrea Clematis, and Daniele D'Agostino. Managing networks of mobiles entities using the hyvonne p2p architecture. In *Proceedings of the 2008 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS '08)*, pages 335–341, Washington, DC, USA, March 4-7 2008. IEEE Computer Society.

[52] C. Zheng, G. Shen, S. Li, and S. Shenker. Distributed segment tree: Support of range query and cover query over dht. In *Proceedings of 5th International Workshop on Peer-to-peer Systems (IPTPS' 06)*, February 27-28 2006.

[53] A. González-Beltrán, P. Milligan, and P. Sage. Range queries over skip tree graphs. *Computer Communications*, 31(2):358–374, 2008.

[54] Rong Zhang, Weining Qian, Aoying Zhou, and Minqi Zhou. An efficient peer-to-peer indexing tree structure for multidimensional data. *Future Generation Computer Systems*, 25(1):77–88, 2009.

[55] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M. Hellerstein, and Scott Shenker. Prefix hash tree: An indexing data structure over distributed hash tables. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing (PODC '04)*, pages 368–368, New York, NY, USA, 2004. ACM Press.

[56] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proceedings of 19th International Conference on Data Engineering (ICDE'03)*, pages 49–60, 5-8 March 2003.

[57] Bin Liu, Wang-Chien Lee, and Dik Lun Lee. Supporting complex multi-dimensional queries in p2p systems. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS '05)*, pages 155–164, Washington, DC, USA, June 10 2005. IEEE Computer Society.

[58] D. A. Tran and T. Nguyen. Hierarchical multidimensional search in peer-to-peer networks. *Computer Communications*, 31(2):346–357, 2008.

[59] Sherif Botros and Steve Waterhouse. Search in jxta and other distributed networks. In *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*, pages 30–35, Washington, DC, USA, August 27-29 2001. IEEE Computer Society.

[60] Li Gong. Jxta: A network programming environment. *IEEE Internet Computing*, 5(3):88–95, 2001.

[61] James Aspnes and Gauri Shah. Skip graphs. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, January 2003.

[62] Ming-Tsung Sun, Chung-Ta King, Wen-Hung Sun, and Chiu-Ping Chang.

Attribute-based overlay network for non-dht structured peer-to-peer lookup. In *Proceedings of the 2007 International Conference on Parallel Processing (ICPP '07)*, page 62, Washington, DC, USA, September 10-14 2007. IEEE Computer Society.

[63] Fabrizio Falchi, Claudio Gennaro, and Pavel Zezula. Nearest neighbor search in metric spaces through content-addressable networks. *Information Processing and Management*, 43(3):665–683, 2007.

[64] Haiying Shen, Ze Li, Ting Li, and Yingwu Zhu. Pird: P2p-based intelligent resource discovery in internet-based distributed systems. In *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS '08)*, pages 858–865, Washington, DC, USA, June 17-20 2008. IEEE Computer Society.

[65] Haiying Shen, Cheng-Zhong Xu, and Guihai Chen. Cycloid: A constant-degree and lookup-efficient p2p overlay network. *Performance Evaluation*, 63(3):195–216, 2006.

[66] Egemen Tanin, Aaron Harwood, and Hanan Samet. Using a distributed quadtree index in peer-to-peer networks. *The VLDB Journal*, 16(2):165–178, 2007.

[67] Roger Zimmermann, Wei-Shinn Ku, and Haojun Wang. Spatial data query support in peer-to-peer systems. In *Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts - (COMPSAC'04)*, pages 82–85, Washington, DC, USA, September 28-30 2004. IEEE Computer Society.

[68] Xinfa Wei and Kaoru Sezaki. Dhr-trees: A distributed multidimensional indexing structure for p2p systems. In *Proceedings of the Proceedings of The Fifth International Symposium on Parallel and Distributed Computing (ISPDC '06)*, pages 281–290, Washington, DC, USA, 2006. IEEE Computer Society.

[69] Adina Crainiceanu, Prakash Linga, Johannes Gehrke, and Jayavel Shanmugasundaram. Querying peer-to-peer networks using p-trees. In *Proceedings of the 7th International Workshop on the Web and Databases (WebDB '04)*, pages 25–30, New York, NY, USA, June 17 - 18 2004. ACM.

[70] Gershon Kedem. The quad-cif tree: A data structure for hierarchical on-line algorithms. In *Proceedings of the 19th conference on Design automation (DAC '82)*, pages 352–357, Piscataway, NJ, USA, 1982. IEEE Press.

[71] Ajit S. Thyagarajan and Stephen E. Deering. Hierarchical distance-vector multicast routing for the mbone. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '95)*, pages 60–66, New York, NY, USA, August 28 - September 01 1995. ACM.

[72] Clay Shields. *Secure Hierarchical Multicast Routing and Multicast Internet Anonymity*. PhD thesis, University of California, June 1998.

[73] G. Agrawal and J. Agrawal. The global multicast routing protocol - a new architecture for hierarchical multicast routing. In *IEEE International Conference on Communications (ICC '03)*, pages 1770–1774. IEEE Press, May 2003.

[74] Tao Xue and Boqin Feng. An efficient and self-configurable publish-subscribe system. In *Proceedings of the Third International Conference Grid and Cooperative Computing (GCC 2004)*, pages 159–163. Springer, 2004.

[75] Zhenhui Shen and Srikanta Tirthapura. Approximate covering detection among content-based subscriptions using space filling curves. In *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS'07)*, page 2, June 2007.

[76] Spyros Voulgaris, Etienne Rivière, Anne-Marie Kermarrec, and Maarten van Steen. Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale

collaborative networks. In *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS'06)*, February 2006.

[77] Peter Triantafillou and Andreas Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (IDCDS'04)*, pages 562–571, 24-26 March 2004.

[78] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[79] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '01)*, pages 11–20. ACM Press, June 2001.

[80] Peter Triantafillou and Ioannis Aekaterinidis. Content-based publish-subscribe over structured P2P networks. In *Proceedings of International Workshop on Distributed Event-based Systems (DEBS'04)*, pages 104–109, May 2004.

[81] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware (Middleware '04)*, pages 254–273, 18-22 October 2004.

[82] Roberto Baldoni, Carlo Marchetti, Antonino Virgillito, and Roman Vitenberg. Content-based publish-subscribe over structured overlay networks. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 437–446. IEEE Computer Society, 6-10 June 2005.

[83] Ioannis Aekaterinidis and Peter Triantafillou. Pastrystrings: A comprehensive content-based publish/subscribe dht network. In *Proceedings of the 26th International Conference on Distributed Computing Systems (ICDCS'06)*, page 23, 2006.

[84] G. Mühl, L. Fiege, and A. P. Bruchmann. Filter similarities in content-based publish/subscribe systems. In *Proc. International Conference on Architecture of Computing Systems (ARCS)*, volume 2299, pages 224–238, 2002.

[85] Peter R. Pietzuch and Jean Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS '03)*, pages 1–8, 2003.

[86] Liping Chen and Gul Agha. State aware data dissemination over structured overlays. In *Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing (P2P '06)*, pages 145–152, 2006.

[87] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Application-level multicast using content-addressable networks. In *Proceedings of the Third International COST264 Workshop on Networked Group Communication (NGC '01)*, pages 14–29, London, UK, 2001. Springer-Verlag.

[88] Roberto Baldoni, Roberto Beraldi, Vivien Quema, Leonardo Querzoni, and Sara Tucci-Piergiovanni. Tera: topic-based event routing for peer-to-peer architectures. In *Proceedings of 1st International Conference on Distributed Event-Based Systems (DEBS '07)*, pages 2–13, 20-22 June 2007.

[89] Spyros Voulgaris, Daniela Gavidia, and Maarten Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.

[90] Weijia Jia, Wanqing Tu, and Jie Wu. Distributed hierarchical multicast tree algo-

rithms for application layer mesh networks. *IEICE Transactions*, 89-D(2):654–662, 2006.

[91] Peter R. Pietzuch and Jean Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS '02)*, pages 611–618, Washington, DC, USA, July 2-5 2002. IEEE Computer Society.

[92] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory (ICDT '99)*, pages 217–235, London, UK, 1999. Springer-Verlag.

[93] Chi-Hoon Lee, Osmar R. Zaïane, Ho-Hyun Park, Jiayuan Huang, and Russell Greiner. Clustering high dimensional data: A graph-based relaxed optimization approach. *Information Sciences: an International Journal*, 178(23):4501–4511, 2008.

[94] Alexander Thomasian, Yue Li, and Lijuan Zhang. Optimal subspace dimensionality for k-nearest-neighbor queries on clustered and dimensionality reduced datasets with svd. *Multimedia Tools and Applications*, 40(2):241–259, 2008.

[95] Haiyong Xie, Y. Richard Yang, Arvind Krishnamurthy, Yanbin Grace Liu, and Abraham Silberschatz. P4p: Provider portal for applications. *ACM SIGCOMM Computer Communication Review*, 38(4):351–362, 2008.

[96] David R. Choffnes and Fabián E. Bustamante. Taming the torrent. a practical approach to reducing cross-isp traffic in peer-to-peer systems. In *Proceedings of the 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '08)*, New York, NY, USA, August 17-22 2008. ACM Press.

[97] O.D. Sahin, S. Antony, D. Agrawal, and A. El Abbadi. PRoBe: Multi-dimensional Range Queries in P2P Networks. In *Proceedings of 6th International Conference on Web Information Systems Engineering (WISE '2005)*, pages 332–346, Munich, Germany, November 20-22 2005. Springer-Verlag Berlin Heidelberg.

[98] Jordi Pujol Ahulló, Pedro García López, Marc Sànchez Artigas, and Antonio F. Gómez Skarmeta. Supporting Geographical Queries onto DHTs. In *Proceedings of 33rd IEEE Conference on Local Computer Networks (LCN'08)*, pages 435–442, 14-17 October 2008.

[99] Minsoo Lee, Yoonsik Uhm, Zion Hwang, Yong Kim, Joohyung Jo, and Sehyun Park. A ubiquitous computing network framework for assisting people in urban areas. In *Proceedings of the 32nd IEEE Conference on Local Computer Networks, 2007 (LCN '07)*, pages 215–216, 15-18 October 2007.

[100] Sylvia Ratnasamy, Brad Karp, Scott Shenker, Deborah Estrin, Ramesh Govindan, Li Yin, and Fang Yu. Data-centric storage in sensornets with ght, a geographic hash table. *Mobile Networks and Applications*, 8(4):427–442, 2003.

[101] Roger Zimmermann et al. Efficient query routing in distributed spatial databases. In *Proc. GIS'04*, pages 176–183, New York, NY, USA, 2004. ACM Press.

[102] Yatin Chawathe, Sriram Ramabhadran, Sylvia Ratnasamy, Anthony Lamarca, Scott Shenker, and Joseph Hellerstein. A case study in building layered dht applications. In *Proc. SIGCOMM '05*, volume 35, pages 97–108, New York, NY, USA, October 2005. ACM Press.

[103] Chi Zhang, Arvind Krishnamurthy, and Randolph Y. Wang. Brushwood: Distributed trees in peer-to-peer systems. In *Proc. IPTPS'05*, volume 3640. Springer, February 2005.

[104] Bivas Mitra, Fernando Peruani, Sujoy Ghose, and Niloy Ganguly. Analyzing

the vulnerability of superpeer networks against attack. In *Proc. CCS '07*, pages 225–234, 2007.

[105] Shuheng Zhou, Gregory R. Ganger, and Peter Steenkiste. Location-based node ids: Enabling explicit locality in dhts. Technical Report CMU-CS-03-171, Carnegie Mellon University, September 2003.

[106] Luis Garces-Erice, Keith W. Ross, Ernst W. Biersack, Pascal A. Felber, and Guillaume Urvoy-Keller. Topology-centric look-up service. In *Proc. of COST264 Fifth International Workshop on Networked Group Communications (NGC)*, Munich, Germany, 2003.

[107] Xiaoping Sun. Scan: A small-world structured p2p overlay for multi-dimensional queries. In *Proc. WWW'07*, pages 1191–1192, New York, NY, USA, May 2007. ACM Press.

[108] Christian and Maihöfer. A survey of geocast routing protocols. *IEEE Communications Surveys & Tutorials*, 6(2), 2004.

[109] Tim Stevens, Joachim Vermeir, Marc De Leenheer, Chris Develder, Filip De Turck, Bart Dhoedt, and Piet Demeester. Distributed service provisioning using stateful anycast communications. In *Proceedings of the 32nd IEEE Conference on Local Computer Networks, 2007 (LCN '07)*, pages 165–174, 15-18 October 2007.

[110] Peter Druschel and Antony Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Proc. HotOS-VIII*, Schloss Elmau, Germany, May 2001.

[111] Jordi Pujol Ahulló, Pedro García López, and Antonio F. Gómez Skarmeta. Towards a lightweight content-based publish/subscribe services for peer-to-peer systems. *Special Issue on Efficient Resource, Service and Data Models for Grid and P2P-Enabled Applications. International Journal of Grid and Utility Computing (IJGUC)*, 1(3):239–251, January 2009.

[112] Gero Mühl, Ludger Fiege, and Peter Pietzuch. *Distributed Event-Based Systems*. Springer-Verlag Berlin Heidelberg New York, 2006.

[113] Jordi Pujol Ahulló, Pedro García López, and Antonio F. Gómez Skarmeta. LightPS: Lightweight Content-based Publish/Subscribe for Peer-to-Peer Systems. In *Proceedings of 2nd International Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC-2008), held in conjunction with International Conference on Complex, Intelligent and Software Intensive Systems (CISIS-2008)*, pages 342–347, Los Alamitos, CA, USA, March 4-7 2008. IEEE Computer Society.