

# Attribute-Based Access to Distributed Data over P2P Networks

Divyakant Agrawal, Amr El Abbadi, and Subhash Suri

Dept. of Computer Science, UCSB, Santa Barbara, CA 93106, USA  
{[agrawal](mailto:agrawal@cs.ucsb.edu), [amr](mailto:amr@cs.ucsb.edu), [suri](mailto:suri@cs.ucsb.edu)}@cs.ucsb.edu  
<http://www.cs.ucsb.edu/~agrawal>  
<http://www.cs.ucsb.edu/~amr>, <http://www.cs.ucsb.edu/~suri>

**Abstract.** Peer-to-peer (P2P) networks are distributed data sharing systems with no dedicated and centralized infrastructure. These systems are attractive because they deliver on the Internet's promise of true decentralization, offering scalability, availability, fault tolerance, robustness, and low barriers to entry. While P2P systems have been used so far mainly for file sharing, their true potential lies as a vast, loosely connected world wide infrastructure for sharing resources, data and information. However, many challenging research problems must be addressed and solved before this vision can materialize. This paper addresses a natural step in the evolution of P2P: going beyond simple file sharing based on exact-name based lookups to data and information sharing where data is accessed based on its attributes or properties. We have identified diverse applications such as network monitoring, astronomy data applications, event-notification systems, and Grid computing that can benefit directly from attribute-based access to distributed data over P2P systems. Based on the application requirements, we propose three new models for both data distribution and data accesses. For each of these models, we propose CAN and CHORD like structures for data storage and information retrieval. A novel aspect of our development is that one of the models identified is directly applicable for building *content-based publish/subscribe* systems over P2P networks.

## 1 Introduction

Peer to peer (P2P) computing has attracted enormous interest recently, both from the commercial and the academic communities. The underlying principle of P2P systems is very simple. A user wishing to participate in a P2P system registers his/her machine and, once registered, becomes a peer node. When a user wants to search for a file, he submits a query string (name of a file), and the system returns the name of one or more peers that contain the file (if it is available in the system). Napster became an overnight sensation as millions of users found it useful to share their music files. Its centralized index, however, was technically deficient and not designed to scale to the large population that it found itself serving. Soon thereafter, other more decentralized file-sharing

systems like Gnutella and Freenet [7] came along that eliminated the need for a centralized index. The popularity of P2P systems has also resulted in several research projects [20, 9, 21, 8, 10, 13, 1, 26, 24, 23, 27] addressing issues such as scalability, fault-tolerance, and security.

In their current form, however, P2P systems are still primarily used for sharing files (or media objects). Yet they possess the potential to become much more than file sharing systems. A grand vision of P2P computing is to combine all the informational resources of the world wide web (data, storage, computing power) into a loosely connected but highly available, reliable, robust system [11]. We envision P2P systems to evolve into something far more significant than a simple file sharing infrastructure. In particular, we identify a natural step in this progressive evolution: moving from accessing files by name to accessing data based on its attributes or properties [15, 17, 16]. In order to motivate this vision better, let us first consider some examples:

1. Internet routers maintain extensive packet logs of network traffic, which are used or monitor network conditions, to detect traffic anomalies, or plan network capacity. A telecommunication network manager may want to query this data: *how much TCP data went across a backbone router  $R$  that was sent from a subnet  $S$  to subnet  $D$  between 1 and 4 PM yesterday?*
2. The World Wide Telescope is an ambitious project to link several large telescopes and make their data repositories, such as the Sloan Digital Sky Survey (SDSS) [28], available to the global community of scientists and scholars. An astrophysicist may wish to query: *find all objects with brightness  $\geq \beta$  in the color spectrum range  $\gamma$  and positional range (right ascension, declination)  $R$ .*
3. One of the key data processing challenges facing the homeland security is to link together the multitude of databases at various airports and install continuous queries like: *alert FBI whenever a new arrival fits one of the given profiles.*
4. The grid computing [12] initiative aims to harness a large number of loosely coupled computers (or storage devices) into a single “transparent” supercomputer. This distributed supercomputer requires a “directory service” that lets users discover what resources are available. For instance, a user may want to know: *which computers in a certain IP address space have at least 2 GHz processor, at least 1 GB memory, and run Linux version  $X$  or later.*

An important theme common to all these examples is that data is distributed and is often too large to reside in a single place. For example, the Sloan Digital Sky database alone contains more than 200 million objects, and each year 5 TB of new raw data is collected. Similarly, Internet routers maintain logs for millions of packets per second. Instead, the P2P paradigm can be used to access data and execute queries obviating the need to consolidate the information at a central repository. In addition, in nearly all of these applications, users also form a natural community of common interest, making P2P architectures quite attractive for managing data. The main observation that can be made from the above examples is that in all cases user queries need to be executed over data

that is potentially distributed over multiple repositories. A P2P paradigm can be used here for accessing and executing such queries obviating the need for consolidating the data and information at a central repository. Unfortunately, current developments in P2P computing, in general, have focussed on providing name-based access to distributed objects. As can be seen from the above examples, in order to retrieve information relevant to these queries, we need to extend the P2P paradigm so that data can be accessed based on its attributes or properties.

Even within the four applications mentioned, there are subtle but significant differences in the nature of data and queries. Another observation that can be made from the above set of examples is that the nature in which these queries access distributed data varies significantly from each other. For instance, in examples 1 and 2, a large amounts of *raw data* are collected, and a wide variety of users ask queries. Such data is likely to be stored for a long time (astrophysics data) or at least days or weeks (network traffic). Example 3, on the other hand, deals with a model where users may pre-install queries (FBI profiles) and new data generates notification events. Thus, queries may be posed *even before data arrive*. Example 4 may be a mixture of these two models: a user may ask to be notified when a certain combination of resources become available, or ask queries about the current availability of resources. One can envision many other potential applications requiring this form of query functionality where data and objects are accessed based on their attributes instead of being accessed by name. For example, applications involving sensor networks where data is constantly generated by geographically dispersed sensors can benefit from attribute-based access to data over P2P systems [18]. Another class of applications is specialized data analysis where data and derived data products are generated in distributed manner. Examples of such applications include weather and climate pattern analysis, demographic information analysis, and geological data pattern analysis. In summary, there is a large number of applications that can benefit from attribute-based data access over P2P systems.

In this paper, we envision P2P data sharing architectures that can be deployed for attribute-based access to distributed data. We start by identifying four different models of data distribution and data access in distributed systems in Section 2. The first model under this classification corresponds to the current exact name-based lookup functionality of P2P systems. In the remaining sections we outline research challenges in the context of the remaining three models identified in Section 2. The paper concludes with a research vision in the context of these models.

## 2 Data Distribution and Attribute-Based Access Models

The P2P storage model is a compelling one for the applications we have suggested. However, as mentioned earlier, data accesses in current P2P systems are largely limited to “name-based exact match” queries, which is inadequate for the kind of applications identified in the introduction. There remains a large

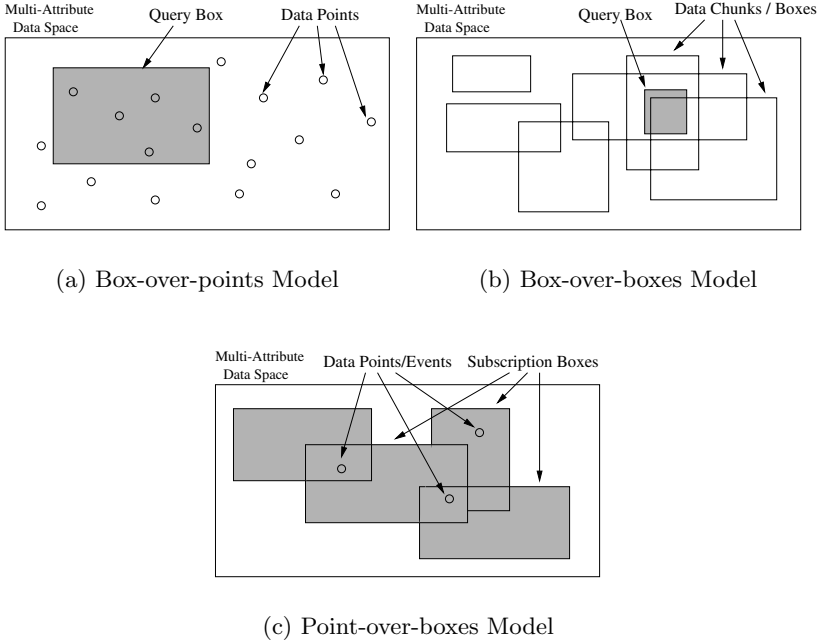
gap between the name-based exact lookup functionality of current P2P systems and the complex attribute-based access functionality of even the most primitive database systems. In this paper, we design and develop P2P architectures in which users can access data based on its properties or *attributes*. A key step in this direction is to use the *relational model* in which data is retrieved from storage systems based on *selection predicates*. Typically, these selection predicates are formulated either as *point queries*, where the exact value of attribute(s) is specified, or as *range queries*, where the attribute value ranges are specified. In general, selection predicates are used by the database engine in conjunction with index and access structures to efficiently retrieve relevant data. Even in the context of complex database operations involving SQL queries, selection operations are the most fundamental operations. Our goal, therefore, is to develop P2P architectures and access mechanisms that will enable efficient execution of selection queries over distributed data.

There are four natural models for access to distributed data. The first one, which we call *point-on-point*, is the basic exact match query: the access is at the level of individual objects or files. The user specifies all necessary attributes of the desired object, and the system determines which, if any, peer contains that object. This is equivalent to the name-based “point” lookup, which is currently available in P2P.

The second model, which we call *box-over-points*, is the basic range query model (Figure 1(a)). A user’s query is a range, which can be thought of as a multi-dimensional *box*, and the system returns all the objects (points) that match this query. This model assumes that the objects of interests are scattered over multiple peers. While such a fine-grained distribution of data may not be attractive or feasible for all applications, there are some cases where this model is a natural fit. Sensor networks, for instance, offer a good example: each sensor acts like the source of a single data value, and typical user queries are ranges, for instance, asking for “average temperature within a geographical region,” “all locations where temperature exceeds a certain value” etc.

Next, we consider the *box-over-boxes* access model, where we assume that *subsets* of the objects are distributed among the peers, and users submit range queries (Figure 1(b)). This model is motivated from the observation that points based object distribution is too fine-grained and therefore is applicable to few specific applications such as sensor data networks. In general, we envision that a large number of application will group objects together based on certain properties and hence the need for box based object distribution. Each peer stores a set of boxes, representing data in certain ranges, and given a query box, the goal is to retrieve enough stored boxes to be able to answer the query. The data stored at peers could be either carefully chosen “chunks” of the source data, or cached copies of answers to previously asked range queries. In either case, the scattering of data is at a much higher level than individual tuples. This model significantly generalizes the box-over-points model, and it is one of the main focus points of our work.

Finally, we consider the *point-over-boxes* model, where peers store boxes, representing ranges, and the query processing involves finding all boxes that



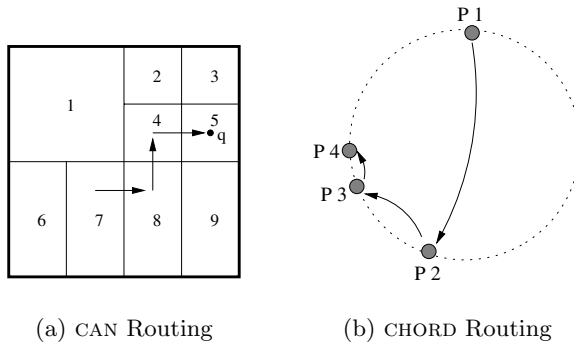
**Fig. 1.** Data Distribution and Attribute-based Access Models

match a point (Figure 1(c)). This model is motivated by the emerging “publish/subscribe” model of data service, where interest profiles of users are modeled as ranges (boxes) in an attribute space, and when an event (point) occurs, the goal is to quickly identify all ranges that contain the point.

### 3 Box-over-Points Distribution and Access Model

One of the main challenges in a P2P system is providing efficient, fault-tolerant, and scalable lookup operations. Early P2P systems either used a centralized [22] or a flooding approach [14]. Although widely used, these solutions suffer from the scalability problem. Either the centralized site is overloaded, or the system is flooded with messages. Several academic efforts are underway to design structured P2P systems to ensure scalable performance as the number of peers and the number of data objects increase. These systems typically use Distributed Hash Tables (DHT) [3] to uniformly distribute the location information about the objects among all the peers. Furthermore, they typically ensure that lookup operations locate the desired objects in a bounded number of hops.

Two of the early and most popular structured P2P systems are CAN [23] and CHORD [27]. We will use these two systems as representative examples for structured P2P systems and will illustrate our extensibility models within these frameworks. In these systems, both the peer identifiers (i.e., IP address) and the



**Fig. 2.** Routing Schemes in Structured P2P Systems

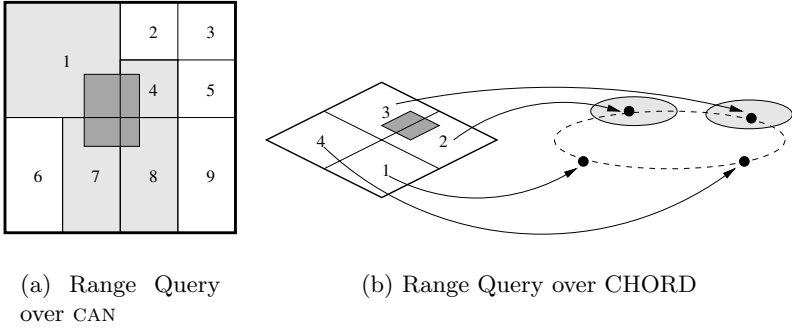
object identifiers (e.g., song name) are uniformly hashed into the same identifier address space. Typically, the hash function is LSA-1, which uniformly hashes the identifier to a 128 bit space.

In CAN [23], the identifier address space is a  $d$ -dimensional space. This space is partitioned into *zones*, each *owned* by a peer. Initially, the entire space is a single zone owned by some initial peer. As nodes join the system, they are uniformly hashed to a point in the multi-dimensional space. The system then splits the zone in which this point is located with the current owner. Likewise, the name or identifier of each new object is hashed into the multidimensional CAN space and the owner of the zone with this point either stores the new object or, more realistically, stores the IP address of the peer storing the object. Lookup simply involves hashing the query identifier to a point in the CAN space and locating the owner of the corresponding zone, who is responsible for returning the requested data, or a pointer to the peer where the requested data is stored as shown in Figure 2(a).

In CHORD [27], the identifier space is a ring, with  $2^{128}$  locations. When a peer joins the system, it is hashed to one of these locations. A new object identifier is also hashed to this circular ring, and the file, or a pointer to the file is stored at the closest peer following the hash point. Lookup simply involves hashing the query identifier to a point in the CHORD ring, and checking the peer following this point (Figure 2(b)).

Since the distribution and access model for current P2P systems fall under the point-on-point model, both CAN and CHORD only support exact match queries, e.g., find the album entitled “Blood on the Tracks”. However, we are interested in more complex *box-over-points* queries, e.g., find all songs by Bob Dylan from 1975–1979. Both of these systems (as described above) would face scalability problems in that they would entail breaking a single range query to an enumeration of multiple point queries. We, therefore, now propose simple and preliminary extensions for adapting these systems to support box-over-points queries.

Although the original CAN proposal associated no semantics with various dimensions, it is relatively straightforward to associate each dimension with one of the attributes or properties of the domain of discourse. For example, one dimen-



**Fig. 3.** Adapting Structured P2P Systems for Range Query Processing

sion corresponds to names (alphabetically ordered) and another corresponds to dates of publication (numerically ordered). Consider an application where each object is associated with  $d$  attributes. We create a  $d$ -dimensional CAN structure. Each new object is mapped to a point in the space corresponding to the values associated with its  $d$  attributes (we assume every point has all attribute values completely specified). The rest of the CAN structure, and methods for peer joins and departures do not change. Now consider a box query with a range in each of the  $d$  dimensions (if some dimension is not specified, we assume the range to be the entire domain). We need to locate all zones that overlap with this box since they may contain some data objects of interest to the query. One simple method to implement this would involve locating the zone corresponding to the bottom left corner (in  $d$  dimensions) of the query box (see Figure 3(a)). The peer owning this zone identifies all objects that were hashed to points in the intersection of the zone and the query box. These form part of the answer set which is returned to the client. However, the peer needs to also propagate the box query to all zones above it and to the right (in the multidimensional space). The query box is thus recursively propagated in this way until it reaches zones which do not overlap with the query.

Alternatively, we could consider CHORD as the underlying P2P structure to support box-over-points queries. In this case we need to map the  $d$ -dimensional attribute space to the one dimensional CHORD ring. One simple approach would involve dividing the  $d$ -dimensional attribute space into *partitions* (see Figure 3(b)). These partitions may be predefined based on the domain range or based on the actual data distribution. Each partition is given a unique identifier; this could be arbitrarily chosen or could be based on, say, the lower left coordinate of the partition. This identifier is used by CHORD to hash the partition to a single peer in the CHORD ring. This peer is now responsible for all the data points which are located in that partition. When a box query needs to be evaluated, the querying peer uses CHORD to locate the peers that own all partitions which overlap with the box query. These peers collectively either store or have pointers to the locations of all data points in the query box.

Although simple and in some way naive, the box-over-points model has many applications. In fact in sensor networks, alternative solutions have been proposed [19]. One main challenge that arises in the context of real applications is the fact that data as well as accesses to the data tend to have non-uniform distribution or equivalently are skewed. This can potentially lead to some peers being responsible for most of the data. Analogously, due to skewed accesses, some peers will have a higher burden for processing queries.

## 4 Box-over-Boxes Distribution and Access Model

We now consider a data distribution and attribute-based data access model in which data is appropriately “chunked” or partitioned and is stored over multiple peers. The reason such “chunking” or partitioning may arise is either due to the physical organization of the system or perhaps as a result of *caching* prior range queries at the peers. For example, in the context of sensor networks, the “chunking” may arise as a consequence of fine-grained readings from individual sensors being collected at a base station responsible for a group of sensors. Similarly, in the case of astronomy data, the partitioning may result from different repositories being responsible for different parts of the sky. We refer to such repartitions in a  $d$ -dimensional space as  $d$ -boxes. Hence the main problem under the box-over-boxes model is that whenever a new query is issued, the peers are searched to determine if the query can be answered from the data boxes stored at the peers. Our goal is to develop techniques that will enable efficient evaluation of range queries over  $d$ -boxes that are distributed (and perhaps replicated) over the peers in a peer-to-peer system. The problem of performing range queries in a peer-to-peer system has recently been investigated [2, 16]. Andrzejak and Xu [2] use Hilbert curves to partition data among peers in a way that contiguously distributes the data at the peers. Gupta et al. [16], on the other hand, use locality sensitive hashing to retrieve  $d$ -boxes “similar” to a query box. Both these approaches, however, are fairly restrictive for supporting box-over-boxes model in full generality.

In order to adhere to the peer-to-peer design methodology, the proposed solution for range lookup should also be based on distributed hashing. A nice property of the DHT-based approach is that the only knowledge that peers need is the function that is used for *hashing*. Once this function is known to a peer, given a lookup request the peer needs to compute the hash value locally and uses it to route the request to a peer that is likely to contain the answer. Given this design goal, a naive approach would be to use a linear hash function over the range query schema  $\langle low, high \rangle$ , i.e., a linear hash function over  $low$ ,  $high$ , or both  $low$  and  $high$ . A simple analysis reveals that such a hash function will enable *only* the exact matches of given range requests, but not set containment matches. As an example, suppose the range  $\langle 20, 35 \rangle$  for a given attribute is stored at a peer. While the query  $\langle 25, 30 \rangle$  does not have an exact match with the stored range, it can be answered using the stored range: we can extract the objects relevant to the query from the superset range  $\langle 20, 35 \rangle$ . The set containment

query functionality is powerful, but harder to provide. We now propose two designs design to solve the range containment problem. The first one utilizes a CAN like structure over the peers and the second one uses the CHORD structure.

#### 4.1 CAN-Based Design

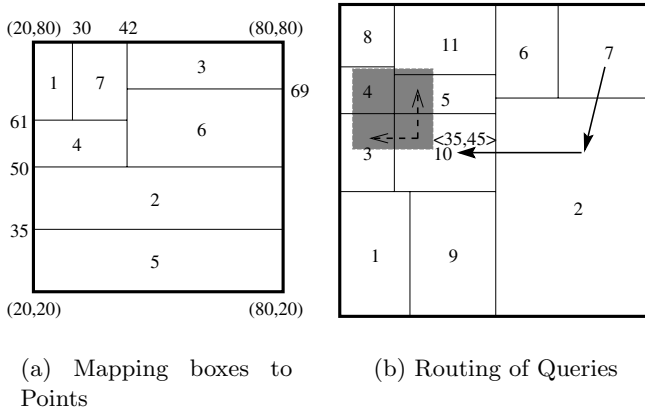
Consider a set of data objects each specified with  $d$  attributes. The main challenge is to map  $d$ -boxes representing a subset of these objects into a CAN-like address space. Rather than associating each dimension of CAN with a single attribute as discussed in Section 3, we associate two dimensions for each attribute. The first dimension captures the start value of the range specified for the attribute, while the second dimension captures the end value of that range. Hence, we use a  $2 \cdot d$ -dimensional CAN and refer to it as 2CAN. A box, hence, is mapped as a single point in this space. Given the domain  $[a, b]$  of an attribute, the corresponding virtual hash space is a two dimensional square bounded by the coordinates  $(a, a)$ ,  $(b, a)$ ,  $(b, b)$ , and  $(a, b)$  in the Cartesian coordinate space. Figure 4(a) shows the corresponding virtual hash space for a range attribute whose domain is  $[20, 80]$ . The corners of the virtual space are  $(20, 20)$ ,  $(80, 20)$ ,  $(80, 80)$ , and  $(20, 80)$ . A range  $\langle q_s, q_e \rangle$  is hashed to point  $\langle q_s, q_e \rangle$  in the virtual hash space.

The virtual hash space is partitioned into zones as was described in Section 3. In this case, a zone can be identified by a pair  $\langle (x_1, y_1), (x_2, y_2) \rangle$  where  $(x_1, y_1)$  is the bottom left corner coordinates whereas  $(x_2, y_2)$  is the top right corner coordinates. Figure 4(a) shows the partitioning of the virtual space. The virtual space is partitioned into 7 zones : *zone-1*  $\langle (20, 61), (30, 80) \rangle$ , *zone-2*  $\langle (20, 35), (80, 50) \rangle$ , *zone-3*  $\langle (42, 69), (80, 80) \rangle$ , *zone-4*  $\langle (20, 50), (42, 61) \rangle$ , *zone-5*  $\langle (20, 20), (80, 35) \rangle$ , *zone-6*  $\langle (42, 50), (80, 69) \rangle$ , and *zone-7*  $\langle (30, 61), (42, 80) \rangle$ . Each zone is assigned to a peer in the system.

For the purpose of routing requests in the system, each peer maintains a *routing table* with the IP addresses and zone coordinates of its neighbors, which are the owners of adjacent zones in the virtual hash space. For example in Figure 4(a), the routing table of the owner of *zone-4* contains information about its four neighbors: *zone-1*, *zone-7*, *zone-6* and *zone-2*.

In the following, for simplicity of exposition, we assume that  $d$ -boxes are generated as a result of caching answers to prior range queries. Alternatively,  $d$ -boxes stored at different zones may be generated via a separate mechanism determined by the underlying application. The point to which a query is mapped to is referred to as the *target point* of the query range. The target point is used to determine where to store the information about the answer of a range query as well as where to initiate range lookups when searching for the result of a range query. The zone in which the target point lies and the node that owns this zone are called the *target zone* and the *target node*, respectively. Therefore, the information about the answer of each range query is stored at the target node of this range.

Initially, the peers in the system may not be populated and therefore a range query must retrieve the results from some initial source of information, e.g., the SDSS repositories. In general, the answer is the  $d$ -box. Once a peer node



**Fig. 4.** Range Query Processing using CAN as the underlying P2P structure

retrieves the answer for its range query, if the peer is willing to share its computed answer and has available storage space, it caches the  $d$ -box and informs the corresponding target node about it. The target node stores a pointer to this querying node. If the target node has available storage, it caches the result itself. In either case, we say that the target node stores the result of this query. For example, according to Figure 4(a), the range query  $\langle 50, 60 \rangle$  is hashed into *zone-6*, so the set of objects that form the answer to this query may be stored at the node that owns *zone-6* or the node will store a pointer to the peer that caches the objects in that range.

When searching for the answer of a range query, the first place to look for cached results is the target zone of this range. Therefore whenever a range query is issued, it is routed toward its target zone through the virtual space. Starting from the requesting zone, each zone passes the query to an adjacent zone until it reaches its target zone. The target node is checked to determine if it stores a  $d$ -box that contains the results of the range query. If so, the result is returned to the client. However, if no such  $d$ -box is found the search is forwarded to all zones that may contain a superset of the answer set. Given the simple geometric properties of 2CAN, such zones are restricted to the upper-left region of the target zone. This is easy to observe since such zones contain  $d$ -boxes with earlier starting values and later ending values than the target zone. Figure 4(b) shows how a query is routed in the system. The range query  $\langle 35, 45 \rangle$  is initiated at *zone-7* and then routed through *zone-6* to its target zone, *zone-10*. If *zone-10* does not contain any  $d$ -box  $\langle X, Y \rangle$  such that  $X \leq 35$  and  $45 \leq Y$  then the search is forwarded to the peers that correspond to zones to the upper left side of *zone-10* since if the answer exists it must be restricted in that part of the attribute space. The search region is illustrated as the shaded region in Figure 4(b).

An important challenge in the design of 2CAN is to limit the length of the routing path and to reduce the space and time overheads of retrieving a qualifying  $d$ -box. An estimation of the average routing distance for processing range

queries in the proposed model is presented in [25]. The analysis shows that the average routing path length in an equally partitioned hash space is  $O(\sqrt{n})$ , where  $n$  is the number of zones in the system. The above analysis easily generalizes to  $2d$ -dimensional space representation. In addition, we have developed techniques for zone maintenance for a dynamic environment where peers are allowed to join and depart from the system freely and frequently. As discussed above, 2CAN results in directed flooding if a query cannot be processed at its target zone. Although this flooding is controlled, a naive approach can give rise to many duplicate messages. We have developed techniques to minimize the duplicate messages when query requests are forwarded. Another aspect of 2CAN is that the upper-left corner of the attribute space may need to bear a greater burden of answering queries. We have also developed load-balancing strategies based on *zone replication* (instead of zone partitioning) to distribute this load.

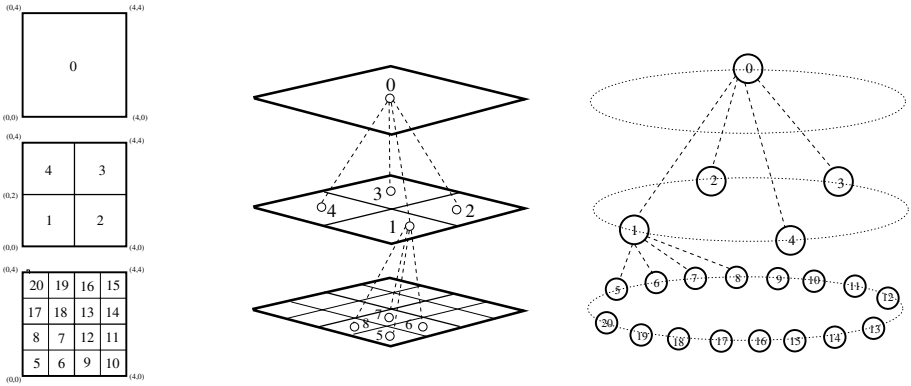
## 4.2 CHORD-Based Design

Our second approach to support box-over-boxes model leverages the  $O(\log n)$  search complexity of CHORD and has two components: a *spatial structure* that organizes the underlying multidimensional attribute space, and a *mapping rule* that associates each  $d$ -box with a unique region of the spatial structure. Let the  $d$ -boxes be in a multidimensional hypercube  $\mathcal{D}$  of side length  $L$ , called the *domain*<sup>1</sup>. We could use the simple  $d$ -dimensional partitioning approach used in Section 3 for point data. However, with  $d$ -boxes the mapping is not as straightforward as with point data. Furthermore, since  $d$ -boxes have different extents, a uniform partitioning may not be efficient. We therefore propose a hierarchical partitioning. We subdivide  $\mathcal{D}$  into  $2^d$  identical boxes, which are the *regions* at *level 1*. Each of these  $2^d$  boxes is then recursively subdivided, until we reach *unit size boxes*.<sup>2</sup> It is easy to see that all the regions are also  $d$ -dimensional hypercubes.

The recursive partition of  $\mathcal{D}$  is naturally modeled by a tree, called the *partition tree*, where each node corresponds to a region produced during the partition. The root (level zero) corresponds to the entire domain  $\mathcal{D}$ . The  $2^d$  children of the root correspond to  $2^d$  boxes of level 1, and so on. Figure 5 shows an example of the spatial partition and its corresponding partition tree. The partition tree defines a parent-child relationship among different regions. With each node, we associate an *id* between 0 and  $\eta - 1$ , where  $\eta$  is total number of regions. The root node has *id* 0. If a node's *id* is  $i$ , then the *ids* of its children are  $i2^d + 1, i2^d + 2, \dots, (i + 1)2^d$ . Call two regions  $r$  and  $r'$  *neighbors* if they are at the same level of the partition tree and are *spatially adjacent*; that is, the two

<sup>1</sup> We choose a hypercube for convenience; with a careful modification, the scheme extends also to rectangular domains.

<sup>2</sup> The unit of space resolution is a user defined parameter. For simplicity, we assume that the original domain is a box of integer dimension  $L$ . Thus, the recursive partitioning stops after  $\log_2 L$  levels.



**Fig. 5.** CONE's Overlay Structure

regions have at least one point in common. In Figure 5, for instance, regions 5 and 7 are neighbors but 5 and 13 are not.

The main challenge now is to map each  $d$ -box to the different regions of the partition tree. This mapping depends on two attributes of a  $d$ -box: a *reference point*, and a *measure*. The reference point can be any consistently defined point in the box. In our scheme, we use the *lower left* corner. More precisely, the reference point of a  $d$ -box  $[x_1, x'_1] \times [x_2, x'_2] \times \dots \times [x_d, x'_d]$ , is the corner  $(x_1, x_2, \dots, x_d)$ . In Figure 5, for example, the reference point of region 5 is the point  $(0, 0)$ . Note that a reference point can be mapped to a region at each of the different levels in the partition tree, e.g.,  $(0, 0)$  in the regions 5, 1, and 0. In order to determine which region a  $d$ -box  $B$  is mapped to, we define the *reference family* of  $B$ , denoted  $\mathcal{F}(B)$ , to be the set of regions in the partition tree that contain the reference point of  $B$ . It is easy to see that  $\mathcal{F}(B)$  contains exactly one region from each level of the partition tree and that the region at level  $i$  is the parent of the region at level  $i + 1$ . Thus, the nodes corresponding to regions of  $\mathcal{F}(B)$  lie on a single path in the partition tree. This scheme will store  $B$  at one of the regions of  $\mathcal{F}(B)$ , but that choice depends on the second attribute: the measure of  $B$ .

The *measure* of  $B$ , denoted  $meas(B)$ , captures the *size* of  $B$ . There are many natural ways to define the measure: the volume, the longest dimension, the shortest dimension, the average dimension, etc. In our scheme, we use *the longest dimension* as the measure. That is,  $meas(B) = \max_{1 \leq i \leq d} |x_i - x'_i|$ . Our *mapping rule* is as follows: A  $d$ -box  $B$  is mapped (uniquely) to the highest region of its reference family whose level is at least  $\log L - \log(meas(B))$ . For convenience, we use the term *native region* to denote the region to which a box is mapped. Thus, if  $meas(B)$  is in the (semi-open) range  $(2^{m-i-1}, 2^{m-i}]$ , where  $m = \log L$ , then the native region of  $B$  is the region in  $\mathcal{F}(B)$  at level  $i$ . The key property of our mapping rule is that all boxes matching a query  $Q$  can be found by searching only the regions that are neighbors of  $Q$ 's reference family. Since  $\mathcal{F}(Q)$  has at most  $\log L$  regions, and each region has a constant number of

neighbors ( $2^d$  in  $d$ -space), our lookup scheme will be able to answer the query by searching  $O(\log L)$  identifiers. We would like to point out that several other natural definitions of the measure do not lead to efficient search. For instance, mapping rules based on the *volume* or the *shortest dimension* can scatter boxes in irregular ways, making it impossible to find boxes matching a query in a small number of regions.

We now describe a generalization of CHORD, which we call CONE, based on the generic scheme described so far. CONE uses the spatial partition of the previous section to build an efficient overlay structure. It generalizes the CHORD structure developed for box-over-points model in a natural way. CONE groups the  $d$ -boxes by their measure: boxes of small measure are associated with regions near the bottom, while boxes of large measure go to regions near the top of the partition tree. CONE is organized in levels too—the name CONE is suggestive of the *conical* shape of the structure, where each level looks like a ring, with rings getting smaller near the top. CONE consists of  $O(\log L)$  CHORD rings. The number of levels or rings is a tunable parameter. The  $i$ th ring manages the set of regions at level  $i$  in the partition tree. For convenience, we use the *ids* of the regions also as the *ids* in the *identifier space* of CHORD. Thus, the *ids* of the regions at level  $i$  are the *identifier space* for the  $i$ th ring. See Figure 5 for illustration. In CONE’s overlay structure, two identifiers are said to have a parent-child relationship if the corresponding regions have one in the partition tree. Similarly, two identifiers are neighbors if the corresponding regions are neighbors.

A peer in CONE belongs to exactly one chord ring. Each peer has an *id* between 0 and  $\eta - 1$ . The identifier corresponding to the peer’s *id* is referred as its *master id*. A peer *owns* all the identifiers whose *ids* lie between its *id* and its predecessor’s *id*. The predecessor of  $p$  is the closest peer in its ring whose *id* is smaller than  $p$ ’s *id*. A peer’s *zone* is the set of identifiers owned by that peer. In Figure 5, for instance, if peers 15 and 19 are the neighboring peers, then the zone of peer 19 includes the identifiers 15–18.

We use the parent-child relationship among the identifiers of the partition tree to define the parent-child relationship among the peers. Specifically, a peer  $p'$  is called the parent of  $p$  if  $p'$  owns the identifier that is the parent of  $p$ ’s master *id*. We refer to  $p$  as a child of  $p'$ . Clearly, each peer has only one parent peer, but multiple children peers. We call two peers neighbors if they own identifiers that are neighbors.

The distribution of peers across rings is tunable. We use the default rule where peers are assigned to rings in proportion to the ring size. One way to implement this rule is to let a new joining peer choose its *id* uniformly between 0 and  $\eta - 1$ , which ensures that it is assigned to the  $i$ th ring with probability  $2^{-d(m-i)}$ , where  $m = \log L$  is the height of the partition tree. CONE extends the basic pointer structure of CHORD. Each peer maintains two additional types of pointers: *level pointers* and *spatial neighbor pointers*. A peer  $p$  (at level  $i$ ) maintains two level pointers, an *up pointer*, which references the parent of  $p$  (at level  $i - 1$ ), and a *down pointer*, which references a child of  $p$  (at level  $i + 1$ ). The level pointers help the search to navigate across levels in constant time,

and they are critical in reducing the search complexity from  $O(\log n \log L)$  to  $O(\log n + \log L)$ . The second type of pointers reference  $p$ 's spatial neighbors (as defined by our spatial structure). These pointers are needed to efficiently search the spatial neighbors of each peer visited during the search.

A query lookup in CONE is done in two phases. We first determine the native region of the query box  $Q$  (from its reference point and measure), and then find the peer  $p$  who owns this native region. In order to locate  $p$ , we use the up/down pointers to reach the correct ring, and then use CHORD to perform the lookup using the native region's *id*. Having found  $p$ , the second phase then recursively searches the neighbors of  $Q$ 's reference family at higher rings. The first phase of the search takes  $O(\log n)$  steps (dominated by the CHORD search), and the second phase climbs  $O(\log L)$  levels of CONE, but each level takes only a constant time; i.e. the up/down pointers allow us to avoid doing a new CHORD lookup search at each level. Thus, the overall search time to locate a matching box is  $O(\log n + \log L)$ .

The basic design of CONE gives us a sound starting point for an attribute-based distributed data access, but it can be improved and extended in several directions. A natural way to improve the "hit rate" of a range cache system is to locate multiple boxes whose union covers the query. Such a scheme can also improve the "quality" of the answer, because one could find boxes whose union covers the query "more tightly" than a single box, hence minimizing the extraneous data. Since locating each additional box also incurs a search cost, we need to balance the total search against the quality of the answer. Both CAN and CHORD use hash functions to "uniformly" distribute the data (as well as the peers) in the search space. However, in many applications, data are highly skewed, and the uniform hashing is likely to create hot spots. In such a case, a more "data adaptive" hashing is needed, and we plan to investigate such hash functions and partitions. In CONE, for instance, we can adapt the assignment of peers to different chord rings to match the load. Analogously, load-balancing strategies need to be developed to handle the case of skewed accesses, which leads to some peers sharing a greater burden of load to process queries.

## 5 Point-over-Boxes Distribution and Access Model

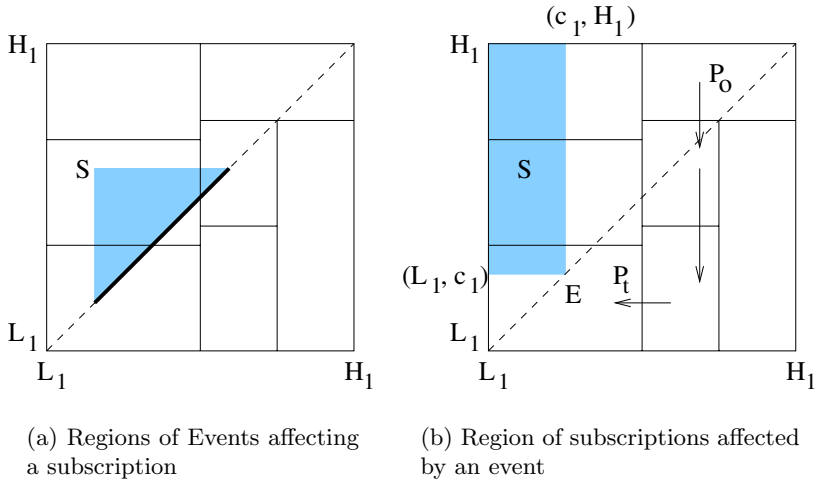
A large class of data access over P2P systems is likely to be in the context of *future data*. Such systems are often referred to as *publish/subscribe* systems. Publish/Subscribe systems are utilized to deliver data events from *publishers* (data/event producers) to *subscribers* (data /event consumers) in a decoupled fashion. Publishers can be completely unaware of the subscribers and simply introduce data events into the system. Subscribers can register their interests with the system in the form of *subscriptions* which act as filters that are used by the system to deliver relevant events to the subscribers. The publish/subscribe system is required to manage the subscriptions and on the occurrence of an event find the matching subscriptions and deliver the event to relevant subscribers.

Existing solutions [5, 4, 6, 29] for publish/subscribe systems employ routing agents for the dissemination of events to the subscribers. Typically, these agents are distributed across the network based on the network locations of the subscribers. These agents form an overlay routing tree which is used to disseminate the events. The agents install filters based on the subscriptions on the overlay links to only send relevant events across. This solution has scalability problems because of the routing bottlenecks at the roots of the routing trees, as well as it is not easy to add more agents to the routing trees.

We employ a peer-to-peer solution for the publish/subscribe system. We utilize the Distributed Hash Table based P2P overlay network for storage of subscriptions and event delivery. This system can be employed directly over the routing agents to improve their scalability. Subscribers can themselves contribute to this system by introducing their machines as peers into the system. A peer-to-peer design for the publish/subscribe system makes it more scalable by utilizing the distributed resources of the agents as well as contributing client machines. In addition, the peer-to-peer design provides the flexibility of adding resources to the system dynamically to scale the system as the demand on the system increases.

We consider a content-based publish/subscribe system with multiple attributes. The schema for the system can be described in the following way:  $\mathbb{S} = \{A_1, A_2, \dots, A_d\}$ , where each  $A_i$  corresponds to an attribute. Each attribute has a name, type and domain, and can be described by the tuple  $\{\text{Name: Type, Min, Max}\}$  where **Min** and **Max** is the domain range of the given attribute. The schema for the publish/subscribe system is known to all the peers participating in the system. A *subscription* is a conjunction of predicates over one or more attributes. Each predicate specifies a constant value (using =) or a range (using  $<$ ,  $>$ ,  $\leq$ ,  $\geq$ ) for an attribute. An attribute cannot appear in more than one predicate. The subscription should specify a continuous range over an attribute  $A_i$ . An example subscription is  $\mathcal{S} = (A_1 \geq v_1) \wedge (v_2 \leq A_3 \leq v_3)$ . An *event* is a set of equalities over all the attributes in the schema. Therefore, an event can be represented as  $\mathcal{E} = \{A_1 = c_1, A_2 = c_2, \dots, A_d = c_d\}$ . Using this nomenclature a subscription can be visualized as a *box* whereas an event corresponds to a *point*. An event  $\mathcal{E}$  matches a subscription  $\mathcal{S}$  if each predicate of  $\mathcal{S}$  is satisfied by the value of the corresponding attribute specified by the event  $\mathcal{E}$ . The publish/subscribe system is required to store the subscriptions specified by the users and given an event, find all subscriptions matching the event and deliver the event to the subscribers. Thus the name *point-over-boxes* model.

We now describe the construction of the logical space used for maintaining the subscription boxes and subsequent routing of event points. We build on the 2CAN P2P system developed for box-over-boxes model in Section 4.1. Given a schema  $\mathbb{S}$  with  $d$  attributes  $\{A_1, A_2, \dots, A_d\}$ , we create a Cartesian space with  $2d$  dimensions. Attribute  $A_i$  with domain range  $[L_i, H_i]$  corresponds to the dimensions  $2i - 1$  and  $2i$  of the Cartesian space. Intuitively, a subscription specifies ranges of interest over the attributes. The starting point and the end point of the range over the  $i$ th attribute can be mapped to the two dimensions



**Fig. 6.** Publish/Subscribe Overlay Structure for Single Attribute Space

corresponding to attribute  $A_i$ . Therefore the domain of the  $2i - 1$  and  $2i$  axes in the Cartesian space is bounded by  $[L_i, H_i]$ . This logical space is partitioned into zones among the peers present in the system, and each peer owns a zone. As was the case in Section 4.1, the peers maintain information about the coordinates of its own zone as well as their neighboring zones. Figure 4(a) in Section 4.1 represents the subscription space for a single attribute.

When a user wishes to subscribe for some events, the user submits the subscription to a peer in the system. We call this peer the *origin* peer  $P_o$ . The origin peer  $P_o$  maps the subscription to its corresponding subscription point in the  $2d$ -dimensional space. The peer whose *zone* contains this point is referred to as *target* peer  $P_t$ .  $P_o$  needs to route the subscription to the target peer  $P_t$ . In order to route the subscription to the target,  $P_o$  selects one of its neighbors, which has the closest Euclidean distance in the  $2d$ -dimensional space to the target point, and forwards the subscription to it. This process of forwarding is continued until the subscription reaches the target zone  $P_t$ . When  $P_t$  receives the subscription, it stores the subscription along with an identifier (e.g., IP address, user name etc.). Figure 4(b) illustrates the routing of subscription  $\langle 35, 45 \rangle$  from *zone-7* to *zone-10*.

When an event is introduced into the system, the publish/subscribe system is required to find all the matching subscriptions installed in the system, and deliver the event to the subscribers. Consider an event  $\mathcal{E} = \{A_1 = c_1, A_2 = c_2, \dots, A_d = c_d\}$ . A subscription  $\mathcal{S} = (l_1 \leq A_1 \leq h_1) \wedge (l_2 \leq A_2 \leq h_2) \wedge \dots \wedge (l_d \leq A_d \leq h_d)$  is affected by event  $\mathcal{E}$  if the following property holds:

$$\forall i \in \{1, 2, \dots, d\} \quad l_i \leq c_i \leq h_i$$

Event  $\mathcal{E}$  is mapped to the point  $\langle c_1, c_1, c_2, c_2, \dots, c_d, c_d \rangle$  in the  $2d$ -dimensional space, and is referred to as an *event point*. The shaded area in Figure 6(a) shows the region of event points in a  $2d$  Cartesian space corresponding to a single attribute schema that can affect the subscription  $\mathcal{S}$ , because all the event points in the shaded region will satisfy the above property. Notice that all events that affect a subscription  $\mathcal{S}$  in the system are located in the bottom right region of the subscription point.

When an event is introduced in the system at a peer  $P_o$  referred to as *origin* peer,  $P_o$  maps the event to its corresponding *event point* and routes the event to the *target* peer  $P_t$  which contains the event point. Figure 6(b) shows the routing path of an event  $E$  and the affected region of  $E$  in a  $2d$  space. The event is then propagated starting from  $P_t$  to all peers which are in the region affected by the event.  $P_t$  sends the event to its immediate neighbors in the affected region, which in turn propagate the event to their neighbors in the affected region. This process continues until all peers in the affected region have been notified of the event.

A particular challenge in P2P systems is the uniform distribution of load among the different peers in the system. Traditional P2P systems are oblivious to the content of the data and hence use a uniform hash function to distribute the data among the different peers. However, in a content-based publish/subscribe system, we distribute the subscriptions and events based on their content. Most real world datasets tend to be skewed and hence will cause a non-uniform distribution of load on the peers. Hence, unlike previous work, we need to use the characteristics of the load to determine how to distribute the load. We have explored alternative approaches. For example, zones that are overloaded due to a heavy subscription load could split their zones with new coming peers. Alternatively zones that are loaded due to heavy event propagation could replicate their information with new peers, thus distributing the event propagation load. These are interesting and novel challenges, which were not studied in earlier P2P systems due to their uniform distribution characteristics. Once, the content of the subscriptions and events is taken into consideration, these issues become crucial for the success of P2P in such data-intensive applications.

Another aspect of 2CAN is that subscriptions with large attribute extents tend to cluster in the upper-left region of the attribute space. In the point-over-boxes model, however, we have the flexibility of representing large subscriptions in terms of subscriptions with smaller extents. Note that this does not cause the end-user any problems since this mapping is completely transparent to the users. In contrast, in the context of box-over-boxes model, this is not possible since breaking-up  $d$ -boxes with large extents will result in either some query boxes not being answered or the query box being flooded to a large number of peers.

The P2P publish/subscribe system can be utilized in critical systems for event monitoring applications, for example power distribution system. The power distribution system consists of *Power Stations* which generate power and send it to *Transmission Substations*. These transmission substations use high voltage

transmission lines to convey power to various *Power Substations* which are located in different geographical areas. The power substations step down the power voltage and distribute it to the residential locations. The power system has sensors that measure the amount of power generated (in MW), transmitted and consumed (in KW) at various points within the system. Sensors also measure the voltage in the transmission lines. Monitoring agents can specify continuous queries that detect anomalies, like sudden drop in voltage over transmission lines or a trip in power generation. This can lead to early detection of problems that can lead to catastrophe. We are currently involved with an energy related company which manufactures hardware for monitoring electrical systems. We plan to work with this company to develop an event notification architecture built on top of these hardware devices which are full-fledged Linux based computing and communication platforms.

## 6 Conclusion

In this paper, we point out that distributed data access at a fundamental level can be classified in four distinct categories. The first one (point-over-point) is the model that has been addressed and solved in existing P2P systems. The second one (box-over-points) represents the basic range searching functionality of traditional database systems. This model can be solved using simple extensions of current P2P systems. However, we argue that this model has certain weaknesses as a general model for distributed data access, and is unlikely to be efficient. We, therefore, propose a natural generalization, the box-over-boxes model, which can handle data distribution at an arbitrary granularity. The box-over-boxes model also allows a mixed use of data replication and caching: the boxes stored at peers can be either data chunks or cached answers to previous range queries. The system tries to find a superset box containing the query box. If no box contains the query box, we can find partial matches. Finally, our point-over-boxes model is a new direction in distributed data service. This is a particularly interesting new model suggested by the new types of “event-notification” services that are emerging around the Internet.

## Acknowledgments

We would like to thank the following students for their contributions to the work described in this paper: Abhishek Gupta, Anshul Kothari, Ozgur D. Sahin, Shyam Anthony, and Chiranjeeb Bourgochain.

We would also like to acknowledge our colleagues Michel Raynal (IRISA, France) and Achour Mostefaoui (IRISA, France) for their feedback on this work.

This work has been supported by the NSF under grants CNF-04-23336, IIS-02-23022, IIS-02-20152, IIS-02-09112, INT-00-95527, and EIA-0080134.

## References

1. Ganesh A., Rowstron A., Castro M., Druschel P., and Wallach D. Security for structured peer-to-peer overlay networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, 2002.
2. Artur Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. In *Proceedings of the 2nd IEEE P2P*, pages 33–40, 2002.
3. Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2):43–48, Feb. 2003.
4. Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Strom, and Daniel C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 262–272, 1999.
5. 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.
6. 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):100–110, 2002.
7. I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, July 2000.
8. Brian Cooper and Hector Garcia-Molina. Peer-to-peer data trading to preserve information. *Information Systems*, 20(2):133–170, 2002.
9. Francisco Matias Cuenca-Acuna, Christopher Peery, Richard P. Martin, and Thu D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. Technical Report DCS-TR-487, Department of Computer Science, Rutgers University, May 2002.
10. Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*, number 2009 in LNCS, pages 67–95, 2000.
11. I. Foster and A. Iamnitchi. On death, taxes, and the convergence of grid and p2p computing. In *Proceedings of the International Workshop on P2P Systems*, 2003.
12. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 1997.
13. Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)*, Washington, D.C., November 2002.
14. Gnutella. <http://gnutella.wego.com/>.
15. Steven Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, and Dan Suciu. What can peer-to-peer do for databases, and vice versa? In *Proceedings of the Fourth International Workshop on the Web and Databases (WebDB 2001)*, Santa Barbara, California, USA, May 2001.
16. A. Gupta, D. Agrawal, and A. El Abbadi. Approximate range selection queries in peer-to-peer systems. In *Proceedings of the First Biennial Conference on Innovative Data Systems Research*, Asilomar, California, January 2003.
17. Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Than Loo, Scott Shenker, and Ion Stoica. Complex queries in DHT-based peer-to-peer networks. In *Proceedings of the first International Workshop on Peer-to-Peer Systems*, 2002.

18. J. Hellerstein. Sensor networks. The Gong Show, First Biennial Conference on Innovative Data Systems Research, 2003.
19. Xin Li, Young Jin Kim, Ramesh Govindan, and Wei Hong. Multi-dimensional range queries in sensor networks. In *Proceedings of the ACM SenSys 2003*, 2003.
20. C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of 16th ACM International Conference on Supercomputing(ICS'02)*, June 2002.
21. Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, 2002.
22. Napster. <http://www.napster.com/>.
23. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.
24. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, pages 329-350, 2001.
25. O. D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. Query processing over peer-to-peer data sharing systems. Technical Report UCSB/TR-2002-28, University of California at Santa Barbara, 2002.
26. Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.
27. 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*, pages 149–160. ACM Press, 2001.
28. Alexander S. Szalay, Jim Gray, Ani Thakar, Peter Z. Kunszt, Tanu Malik, Jordan Raddick, Christopher Stoughton, and Jan vandenBerg. The SDSS skyserver: public access to the Sloan digital sky server data. In *Proceedings of the ACM International Conference on Management of Data*, pages 570–581, 2002.
29. 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*, pages 11–20. ACM Press, 2001.