Distributed Search and Pattern Matching

Reaz Ahmed and Raouf Boutaba

Abstract Peer-to-peer (P2P) technology has triggered a wide range of distributed applications including file-sharing, distributed XML databases, distributed computing, server-less web publishing and networked resource/service sharing. Despite of the diversity in application, these systems share common requirements for searching due to transitory nodes population and content volatility. In such dynamic environment, users do not have the exact information about available resources. Queries are based on partial information. This mandates the search mechanism to be *flexible*. On the other hand, the search mechanism is required to be bandwidth *efficient* to support large networks. Variety of search techniques have been proposed to provide satisfactory solution to the conflicting requirements of search efficiency and flexibility. This chapter highlights the search requirements in large scale distributed systems and the ability of the existing distributed search techniques in satisfying these requirements. Representative search techniques from three application domains, namely, P2P content sharing, service discovery and distributed XML databases, are considered. An abstract problem formulation called Distributed Pattern Matching (DPM) is presented as well. The DPM framework can be used as a common ground for addressing the search problem in these three application domains.

1 Introduction

Peer-to-peer (P2P) technology has triggered a wide range of distributed systems beyond simple file-sharing. Distributed XML databases, distributed computing, server-less web publishing and networked resource/service sharing are only a few to

Reaz Ahmed
Bangladesh University of Engineering and Technlogy, Dhaka, Bangladesh,
e-mail: `reaz@cse.buet.ac.bd`

Raouf Boutaba
University of Waterloo, Ontario, Canada, e-mail: `rboutaba@bbcr.uwaterloo.ca`

name. Despite of the diversity in applications, these systems share a common problem regarding search and discovery of information. This commonality stems from the transitory nodes population and volatile information content in the participating nodes. In such dynamic environment, users are not expected to have the exact information about the available objects in the system. Rather queries are based on partial information, which requires the search mechanism to be flexible. On the other hand, to scale with network size the search mechanism is required to be bandwidth efficient. High levels of content and node dynamism in modern large scale distributed systems, including P2P content sharing, service discovery and P2P databases, impose additional requirements on the search mechanism. Flexibility in query expressiveness and fault-resilience of the search mechanism become more important in such environments.

Since the advent of P2P technology experts from industry and academia have proposed a number of search techniques to provide satisfactory solution to the conflicting requirements of search efficiency and flexibility in distributed environment. This chapter will present the challenges in distributed search and the existing search techniques in three well-known P2P application domains: content sharing, service discovery, and distributed XML databases. A generic formulation, namely Distributed Pattern Matching (DPM) problem, will also be presented. The DPM construct can be used as a generic framework for addressing the search requirements in different P2P applications. Two known solutions to the DPM problem will be highlighted as well.

This chapter is organized as follows. Common properties of large scale distributed systems from three application domains, namely, content sharing, service discovery and distributed XML databases, are presented in Section 2. Desirable characteristics of a distributed search mechanism are presented in Section 3, while the components of a distributed search system are identified in Section 4. Representative search techniques form the above mentioned three application domains are investigated in Sections 5–7, respectively. The DPM abstraction is presented in Section 8. Finally, concluding remarks are placed in Section 9.

2 Large Scale Distributed Systems

Networks of tens or hundreds of thousands of loosely coupled devices have become common in today's world. The interconnection networks can exist in physical or logical dimensions as well as wired and wireless domains. The Internet is the largest distributed system that connects devices through TCP/IP protocol stack. On top of this network there exists many logical overlay topologies, where networked nodes federate to achieve a common goal. Examples of such federations include, the Domain Name resolution System (DNS), the World Wide Web (WWW), content sharing P2P systems, world wide service discovery systems and emerging distributed XML database systems. Among these systems, the WWW and the DNS are mature enough and are characterized by relatively static population of hosts.

Content dynamism is also much lower in these two systems, compared to P2P systems. Centralized and clustered search techniques (e.g., web crawlers and proxy caches) work well for a network of relatively stable hosts (or web sites) or domain name resolvers. Decentralized (control) and distributed (workload) search techniques are required for a network composed of transient populations of nodes having intermittent connectivity and dynamically assigned IP addresses.

Content sharing, service discovery and distributed XML databases are three representative P2P application domains that can be taken as the representatives of modern large scale distributed systems. The identifying properties that separate these application domains from contemporary large scale networks, like WWW and DNS, can be summarized as follows:

Population dynamism: Transient population of nodes mandates the routing mechanism to be adaptive to failures. Redundant routing paths and replication can improve availability and resilience in such environments.

Content dynamism: Frequent arrival of new content, relocation (e.g., transfer) of the existing contents and shorter uptime of peers (compared to internet hosts) are the main causes of content dynamism in these systems. Users in these systems often do not have the exact information (e.g., exact filename, or Service Description) about the content they are willing to discover. Rather most of the queries are partial or inexact, which requires the search mechanism to be flexible.

Heterogeneity: In these systems participating population of nodes display wide variation in capacity, e.g., computing power, network bandwidth and storage. This mandates the index information and routing traffic to be distributed based on nodes' capacities.

2.1 Content Sharing

Content (e.g., file) sharing is the most popular P2P application. A classification of the topologies adopted in various P2P content sharing systems can be found in [20]. In [7], a survey and taxonomy of content sharing P2P systems are presented. All content sharing P2P systems offer mechanisms for content lookup and for content transfer. Although content transfer takes place between two peers, the search mechanism usually involves intermediate entities. To facilitate effective search, an object is associated with an index file that contains the name, location, and sometimes a description (or keywords) of the content. Search for a content typically involves matching a query expression against the index files. P2P systems differ in how this index file is distributed over the peers (architecture) and what index scheme is used (i.e., index structure). From an architectural point of view (see Fig. 1), content sharing P2P systems can be *centralized*, *decentralized*, or *partially-decentralized* [7]. *Centralized* P2P systems are characterized by the existence of a central index server, whose sole task is to maintain the index files and facilitate content search. Napster belongs to this category. Centralized P2P systems are highly effective for partial keyword search, but the index system itself becomes a bottleneck and a single point

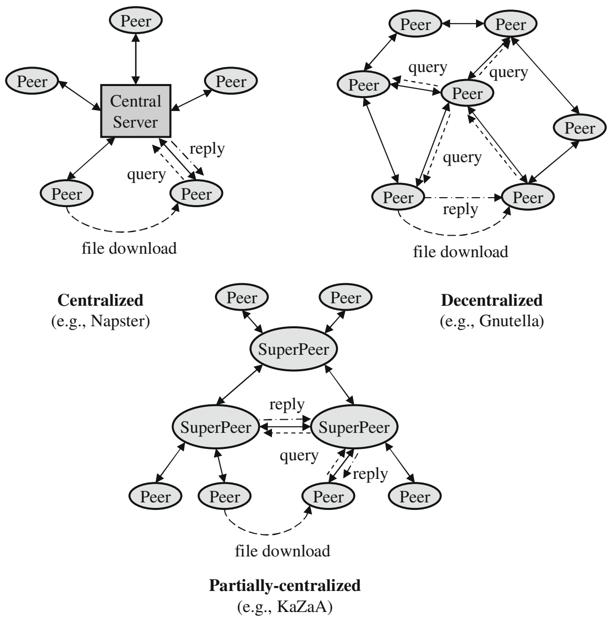


Fig. 1 Content sharing P2P architectures

of failure. *Decentralized* architectures remedy this problem by having all peers index their own local content, or additionally cache the index of their direct neighbors. Content search in this case consists in flooding the P2P network with query messages (e.g.,through TTL-limited broadcast in Gnutella). A decentralized P2P system such as Gnutella is highly robust, but the query routing overhead is overwhelming in large-scale networks. Recognizing the benefit of index servers, many popular P2P systems today use *partially-decentralized* architectures, where a number of peers (called superpeers) assume the role of index servers. In systems such as KaZaA and Morpheus, each superpeer has a set of associated peers. Each superpeer is in charge of maintaining the index file for its peers. Content search is then conducted at the superpeer level, where superpeers may forward query messages to each other using flooding. The selection of superpeers is difficult in such a scheme, as it assumes that some peers in the network have high capacity and are relatively static (i.e., available most of the time). A newer version of Gnutella [71] also uses this approach.

The indexing scheme used by content sharing P2P systems can be categorized as *unstructured*, *semi-structured*, or *structured* [7]. *Unstructured* P2P systems use flat index files, where a index file has no relation to other index files. Napster, Gnutella and KaZaA/Morpheus belong to this category. *Semi-structured* P2P systems, such as Freenet [21] and JXTA [15], use a local routing table at each peer. A search is based on filenames that are hashed to binary keys. The query is routed at each peer to the closest matching key found on the local routing table. To prevent infinite querying, a time-to-live (TTL) value is used. Such mechanism is effective when the content is well replicated over many peers. However, it is virtually impossible to enforce data consistency for file updates. *Structured* P2P systems are specialized in exact matching queries using fully distributed routing structure. Examples of this approach include P2P systems that use distributed indexing and querying schemes, such as Chord [64], CAN (Content Addressable Network) [54] and Tapestry [72].

Advertisements in these systems mostly contain the filename and author-name. For example a movie file can be advertised as *"The Lord of the Rings – The Two Towers – 2002 (Extended Edition) DVDrip.avi"*. For a user it is very unlikely to know the exact name of the advertised file. Rather the user specifies some keywords that may be present in the advertised file name. For example a typical query for the above movie would be *"Lord of the Ring Two Tower"*. Note the keywords *"Ring"* and *"Tower"*; they do not contain the *"s"* as contained in the advertised keywords. This mandates the support for partial keyword matching in P2P content sharing systems.

2.2 Service Discovery

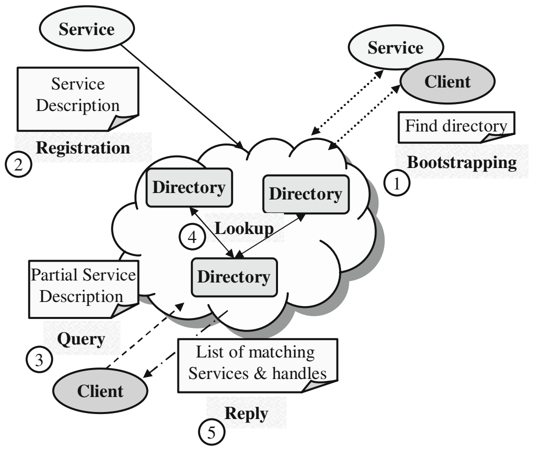


Fig. 2 Service discovery: Generic architecture and steps

Service discovery is an integral part of any service infrastructure. A large-scale service infrastructure requires a service discovery system that is open, scalable, robust and efficient. Most of the service discovery systems rely on a three-party architecture, composed of *client*s, *service*s and *directory* entities. Directory entities gather advertisements from service providers and resolve queries from clients. The generic service discovery mechanism can be viewed as a five-step process (see Fig. 2) [5]; (1) *bootstrapping*, where clients and service providers attempt to initiate the discovery process via establishing the first point of contact within the system, (2) *service advertisement*, where a service provider publishes information (a Service Description containing a list of property-value pairs) to a directory entity about the provided service (3) *querying*, where a client looks for a desired service by submitting a query (usually a partial Service Description) to a directory entity, (4) *lookup*, where the directory entity searches the network of directory entities for all Service Descriptions matching the query and (5) *service handle retrieval*, the final step in the discovery mechanism, where a client receives the means to access the requested service. Some of these steps may be omitted in various discovery approaches. Some of the discovery approaches are based on two-party (client-server) architecture without any directory infrastructure.

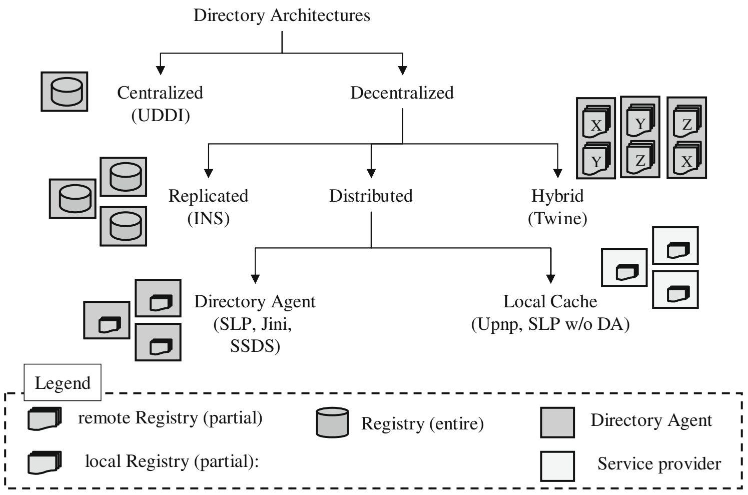


Fig. 3 Taxonomy of the directory architectures

Directory architectures adopted by different service discovery approaches can broadly be classified as *centralized* and *decentralized* (see Fig. 3) [5]. In a centralized architecture, a dedicated directory entity or registry maintains the whole directory information (as in centralized UDDI [69]), and takes care of registering services and answering to queries. In decentralized architectures, the directory information is stored at different network locations. Decentralized systems can be categorized as *replicated*, *distributed* or *hybrid*. In the replicated case, the entire directory information is stored at different directory entities (as in INS [2]). In the distributed case, the directory information is partitioned, and the partitions are either stored in dedicated directory entities (DA) (as in SSDS [24], SLP [30] and Jini [65]) as per a three-party model or cached locally by the service providers in the system (e.g., UPnP [47] and SLP in DA-less mode), according to a two-party model. Finally, in the hybrid case, the system stores multiple copies of the entire directory information without assigning the entire registry to a single directory entity (as in Twine [11]).

In large-scale networks, a centralized directory becomes a performance bottleneck and a single point of failure. Consistency of the replicas is a major issue in the replicated architecture (like INS), since maintaining consistent replicas is usually bandwidth-consuming. On the other hand, when the directory information is distributed, e.g., partitioned among dedicated directory entities, the failure of one of them leads to the unavailability of part of the directory information. The fully distributed two-party architecture attempts to remedy all these issues, however these systems generally do not scale well, since they use multicast-like communications which are expensive in terms of bandwidth. Hybrid architectures seem to offer the best compromise between bandwidth consumption, scalability, and fault-tolerance.

Advertisement	***Query***
Service-type=service:print	**Service-type=service:print**
Scope-list=staff, grad	*Scope-list=grad*
Location=DC3335	*Paper-size=A4*
color=true	
language=PS	
Paper-size= legal, A4, B5	
URL=diamond.uwatreloo.ca/PCL8	

Fig. 4 Example advertisement and query in service discover systems

An example of a generic advertisement and a query in service discovery systems is presented in Fig. 4. In these systems a service is advertised using a list of descriptive property-value pairs, called a *Service Description*. A Service Description typically contains service type (e.g., *Service-type=service:print*), service invocation information (e.g., *URL=diamond.uwatreloo.ca/PCL8*) and service capabilities (e.g., *Paper-size= legal, A4, B5*). In most cases a Service Description is instantiated from a *Service Schema*, which contains meta-information regarding the Service Descriptions for a given class of service (e.g., print service or *service:print*). A Service Schema governs the allowable properties and their types (e.g., string, integer, float, *etc.*) within the Service Descriptions of a given class of services. In most service discovery systems it is assumed that the available Service Schemas are globally known.

Queries in these systems (see Fig. 4) usually contain the requested service type and a list of required capabilities of the service (e.g., Paper-size=A4). The list of capabilities provided in a query is a subset of the capabilities list provided in the advertisements it should match against. The result of a query consists of a list of Service Descriptions matching the query.

2.3 Distributed XML Databases

Distributed XML databases on P2P systems, or P2P Databases Systems (PDBS) in short, have been investigated, more recently, following the success of P2P content sharing. A P2P database system can be thought of as a data sharing network built on top of a P2P overlay. Search in PDBS demands more flexibility than that required by the P2P content sharing systems. This requirement stems from the existence of semantic (schema) information associated with the shared data. Most of the research works focus on building an additional layer on top of the existing P2P search techniques.

Though PDBSs evolved as a natural extension of Distributed Database Systems (DDBS), they have a number of properties that distinguish them from the DDBS and traditional Database Management Systems (DBMS). Unlike DDBS, PDBS has no central naming authority, which results into heterogenous schemas in the system. Due to the absence of any central coordination and the large-scale evolving topology, a peer knows about only a portion of the available schemas and data. This mandates a mechanism (e.g., ontology) for unifying semantically close schemas. In a DDBS, arrival or departure of nodes is performed in a controlled manner, which is not true for PDBSs. Finally, in contrast to DDBS, a peer in a PDBS has full control over its local data.

In PDBS, semantic mapping of schema is a challenging problem. It requires interoperation between heterogenous data models. XML [63] is used as the defacto standard for this purpose. A survey on the use of XML in PDBS can be found in [39]. In PDBS, XML is used in two ways. Firstly, XML is used for representing data and data models (i.e., schema information). Secondly, XML is used to represent semantic relationships among heterogeneous data models at three different levels: schema level, element level and data level. These levels of granularity also influence the indexing mechanism adopted in these systems.

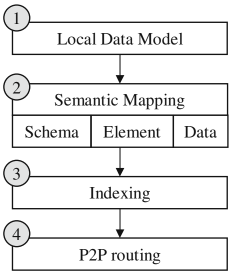
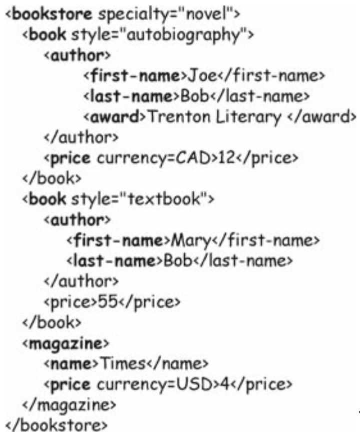


Fig. 5 Functional layers in a PDBS system

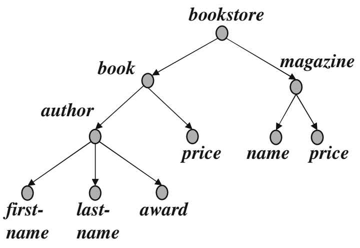
Figure 5 presents the possible functional layers in a PDBS. Each peer in the system has its own local data model independent of the other peers' data models. The process of translating a local query to other peers' data models is performed by the semantic mapping layer at different granularities. The third layer is optional, and can maintain indices at different granularities. Finally, the fourth layer is usually one or a combination of the routing mechanisms present in traditional file sharing P2P systems.

Many research works on PDBS assume the existence of an underlying P2P substrate for efficient and flexible query routing, and concentrate on higher level issues including semantic mapping between heterogenous schemas and distributed query processing and optimization.



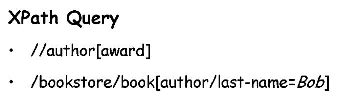
```
<bookstore specialty="novel">
  <book style="autobiography">
    <author>
        <first-name>Joe</first-name>
        <last-name>Bob</last-name>
        <award>Trenton Literary </award>
    </author>
    <price currency=CAD>12</price>
  </book>
  <book style="textbook">
    <author>
      <first-name>Mary</first-name>
      <last-name>Bob</last-name>
    </author>
    <price>55</price>
  </book>
  <magazine>
    <name>Times</name>
    <price currency=USD>4</price>
  </magazine>
</bookstore>
```

(a) An XML advertisement



(b) Tree representation

Fig. 6 Advertisement in PDBS



XPath Query

- //author[award]
- /bookstore/book[author/last-name=*Bob*]

Fig. 7 XPath query examples

Advertisement and query in PDBS are more complicated than that in P2P content sharing and service discovery systems. Figure 6 depicts an example of an XML advertisement which contains information about two books and one magazine. A tree representation of the corresponding *XML Schema* [26] has been presented in Fig. 6b. Analogous to Service Schema, an XML Schema contains meta-information regarding a class of XML documents. However, the syntax used for describing XML documents and XML Schema are standardized and widely used, compared to the variations in Service Description and Service Schema definition syntaxes used by different service discovery systems.

The most popular query syntax used in PDBS is XPath. Figure 7 presents two examples of XPath queries based on the advertisement presented in Fig. 6. The first query finds all *author*s having at least one *award*. The second example finds all books for which *last-name* of the *author* is *Bob*.

3 Distributed Search Requirements

Search is an essential functionality offered by any distributed system. A search mechanism in a distributed system can be either centralized or distributed. For *Centralized Search* there exists a central core of one or more machines responsible for indexing the contents distributed across the network and for responding to user queries. For networks with lesser degree of dynamism, centralized search mechanisms prove to be adequate. Google, Yahoo, Alta vista, *etc.* are the living examples of centralized search mechanisms, where a set of crawlers running on a cluster of computers index the Webpages around the globe. Compared to the lifetime of the contents shared in P2P networks, Webpages are long lived. Centralized search techniques do not prove to be efficient in large scale distributed systems due to content and population dynamism (as explained in Section 2). *Distributed Search* mechanisms assume that both indexing mechanism (analogous to crawlers) and indexed information are distributed across the network. Consequently the design requirements for Distributed Search techniques are different from that for Centralized Search techniques. In the following is a list of the most important design requirements for a Distributed Search mechanism.

- *Decentralization*: For a Distributed Search mechanism to be successful, decentralization of control and data are necessary. Decentralization of control refers to the distribution of the index construction process among the participating nodes. There should not be any central entity governing the index construction process in different nodes. Unlike web search engines, the index itself should be distributed across the participating nodes for achieving uniform load distribution and fault-resilience.
- *Efficiency*: The search mechanism should be able to store and retrieve index information without consuming significant resource: mainly storage and bandwidth. In a large scale distributed system advertisements are frequent due to the arrival of new documents and relocation of existing documents. The large user base generates queries at a high rate. This mandates both advertisement and search process to be bandwidth efficient.
- *Scalability*: Efficiency of the search mechanism should not degrade with increase in network size. In addition the number of links per node should not increase a lot with growth in network size. Join and topology maintenance overhead depends largely on the number of links that a node has to maintain, especially in dynamic environments.
- *Flexibility*: Due to content dynamism, users do not usually have the exact information about the advertised objects. The query semantics offered by the search mechanism should be flexible to support inexact or subset queries. The scalability and efficiency requirements should not be sacrificed for achieving the flexibility requirement.
- *Search completeness*: Search completeness is measured as the percentage of advertised objects (matching the query) that were discovered by the search. Required level of search completeness varies from application to application. A search mechanism should have guarantee on the discovery of rare objects. In the case of popular or highly replicated objects, only a predefined number of matches would suffice for most cases. For specific queries, the number of matching objects would be low and all of them should be discovered by the search. Broad queries, on the other hand, would match a large number of advertised objects. In this case search result may be restricted within a predefined limit to avoid high bandwidth consumption.

- *Fault-resilience*: In large scale distributed systems, participating nodes connect autonomously without administrative intervention. Nodes depart from the network without a priori notification. The search mechanism is expected to advertise and discover objects in a continuously evolving overlay topology, resulting from the frequent arrival and failure of nodes. In many cases index replication and pair-wise, alternate routing paths are used to improve availability.
- *Load distribution*: Heterogeneity in nodes' capabilities, including processing power, storage, bandwidth and uptime, is prominent in large scale distributed systems. To avoid hot spots and to ensure efficiency, the advertisement and search mechanisms should distribute routing, storage and processing loads according to the capabilities of the participating nodes. In other words, uniform distribution of load may result into poor system performance in a large scale distributed system.

In addition to the above mentioned design requirements, a number of other requirements of secondary importance may arise in different scenarios. For example,

- *autonomy* of index placement and routing path selection may be required for security and performance reasons;
- *anonymity* of the advertising, indexing and searching entities may be required in censorship resistance systems;
- *ranking* of search results may be required for full-text search or information retrieval systems; *etc*.

4 Components of a Distributed Search System

In a large scale distributed system, a distributed search mechanism can be composed of three components as depicted in Fig. 8 and presented in following list.

1. *Query semantics* refer to the expressiveness of a query and the allowed level of semantic heterogeneity in queried and advertised information.
2. *Translation* is a function governing the transformation of semantic information present in a query to a form, suitable for query routing.
3. *Routing* refers to the mechanism of forwarding a query to the nodes suitable for answering the query.

Each of these components are explained in greater detail in the following subsections.

4.1 Query Semantics

Any visible (e.g., shared or advertised) object in a distributed system is associated with a set of properties describing the behavioral and functional aspects of that object. Meta information on a set of related properties associated with a class of objects

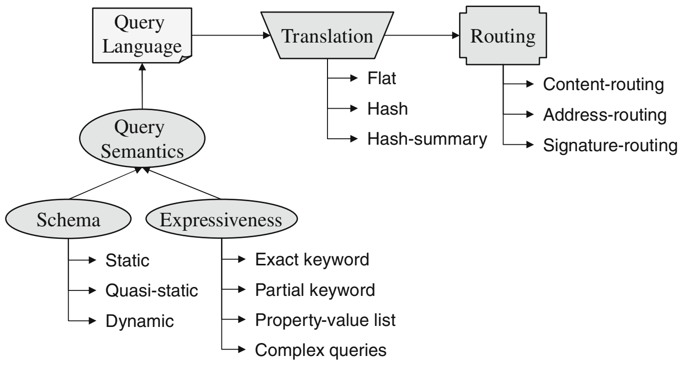


Fig. 8 Components of a distributed search system

is defined as the *schema* for that class of objects. In a distributed search system, structure and scope (temporal and spatial) of the available schemas influence the query language capability and underlying routing mechanism. The rest of this section highlights two aspects of query semantics: *schema* and *query expressiveness*.

4.1.1 Schema

Based on the temporal and spatial scope of the schema, large scale distributed systems can be classified as follows:

- *Static schema*: Most of the file sharing P2P systems have been designed to share one or more specific types of files, e.g., song, movie, software *etc*. For each type of file a specific set of properties is defined that remain unchanged throughout the life of the system. Essentially these systems have one or more static schemas that are globally known.
- *Quasi-static schema*: Most of the service discovery systems fall into this category. Unlike file sharing P2P systems, service discovery systems allow dynamic creation of schema for describing services. Each service instance is advertised as a *Service Description* governed by a predefined *Service Schema* (or template). All schemas in a given service discovery system have to contain a minimal set of predefined *properties* to comply with the specific system under consideration. Though schema can be created dynamically, the rate of such events is very low and the number of available schemas in a given system is much lower than that in PDBS. Furthermore, it is assumed that all the existing schemas in the system are globally known.
- *Dynamic schema*: Most of the PDBSs fall into this category. In these systems heterogeneous schemas exist. Temporal scope of a schema is often bounded by the lifespan of the peer advertising data with that schema. Spatial scope is local to the originating peer and its neighbors; no global knowledge is assumed. In such systems, Automating the process of semantic mapping between similar

schemas is a challenging problem, which may require additional support from the underlying routing mechanism.

4.1.2 Expressiveness

Query expressiveness refers to the capability of the query language in expressing information retrieval requirements. Exiting research works focus on a wide variety of query expressiveness ranging from simple keyword-based queries to complex queries, such as LDAP filter [34] and XPath [38]. Below is a non-exhaustive list of the different levels of query expressiveness commonly found in distributed search techniques.

- *Exact keyword match* is the minimum level of query expressiveness supported by any search mechanism, and is present in most of the file sharing P2P systems, especially the ones based on DHT techniques. For this level of expressiveness, a globally known fixed schema (with a limited number of properties) is assumed.
- *Partial keyword match* is supported by most of the unstructured techniques as well as some extensions to the DHT techniques. Two major variants in this category can be found. Most extensions to DHT techniques support *partial prefix matching* and unstructured techniques support true *partial matching*.
- *Property-value list* is used by many service discovery techniques. Service Descriptions are specified as a property-value list, and queries are specified as a subset of the advertised property-value list. Most service discovery techniques assume a flat list of property-value pairs and do not support wildcard-based partial matching in property names or values.
- *Complex queries* involve logical and relational operators (i.e., *range queries*), and hierarchical relations between properties. Complex queries are supported by a few service discovery approaches and most of the distributed XML database systems. As a means of expressing a distributed query forml query languages, such as LDAP filters, XQuery [18], XPath [38] and SPARQL [51] are used.

4.2 Translation

In most distributed systems the query expression specified by a user is not used *as is* by the underlying routing mechanism. Instead, the query expression goes through some kind of transformation before it is fed to the routing process. The translation function works as a bridge between user specified queries and the routing mechanism. The *domain* of the translation function is governed by the query semantics as discussed in the previous section. The *range* of the translation function, on the other hand, depends on the routing mechanism used by the underlying overlay. Based on the particular combination of query semantics and routing mechanism, this function can exhibit a wide variation. Translation functions can be broadly classified into the following three categories:

- *Flat*: This type of translation functions do a very little (e.g., filtering) or no change to the query expression and associated semantic information. Such functions are usually used by unstructuredand semi-structured routing mechanisms, and most of the industrial approaches to service discovery.
- *Hash* : Hashing is mostly used by structured and semi-structured search mechanisms. A wide variety of hashing techniques have been proposed for distributed search systems. However, the major problem with this type of translation functions is that they loose semantic information during the hash transformation process. As a result only exact or prefix matching is supported by the search mechanisms that adopt hashing as translation function.
- *Hash-summary*: This type of translation enables efficient query routing while preserving query semantics. Variants of Bloom filters are the most popular means of representing hash summaries. Hash summaries are mostly used by unstructured and semi-structured search mechanisms.

4.3 Routing

In overlay networks, routing refers to the process of forwarding a message from a source node to a destination node. The source and the destination nodes are usually at a number of hops away from each other on the overlay. Routing algorithms in overlay networks can be broadly classified into two categories: *uninformed* and *informed*. *Uninformed* routing algorithms do not use the knowledge of query semantics or target node's address in making routing decisions at each hop. *Flooding* [1], *Random walk* [45] and *Iterative deepening* [71] are the representative algorithms in this category. These algorithms are not efficient in terms of the generated volume of search traffic, but the robustness is good in highly dynamic environment. Based on the nature of information used for next hop selection, *Informed* routing algorithms can be classified into the following three categories:

- *Content routing (CR)*: Content routing algorithms utilize the semantic information, embedded in user query, for making routing decisions at each hop. Hence, the associated translation function should be from the *flat* category. Examples of such routing strategy include, selective flooding [22] and hint based routing [68]. Content-routing allows partial match and complex queries, but the offered query routing efficiency is low. Moreover, there exists no guarantee on search completeness or the discovery of rare objects.
- *Address routing (AR)*: Address routing is adopted in DHT-based structured P2P overlays, such as Chord [64], CAN [54], Pastry [57] and Kademlia [46]. Different *hash* techniques are used to transform a query into a virtual address on the overlay, and this address is used to route the query to a responsible node. Routing algorithms in this category are efficient in terms of query routing traffic, but they are not appropriate for semantic laden search (e.g., partial matching and complex queries).

- *Signature routing (SR)*: A number of distributed search techniques, including [3, 24, 42] construct a signature (usually a Bloom filter) of the target object and routes queries based on this signature. These techniques strive to combine the merits of both content-routing and address-routing strategies. Signatures retain (part of or the whole) query semantics and allow aggregation for efficient indexing. However, search completeness and robustness are not as good as that in address-routing and content routing, respectively.

5 Search Techniques in Content Sharing P2P Systems

5.1 Structured Techniques

Majority of the structured search techniques rely on Distributed Hash Tables (DHT). In general DHT-based techniques, like Chord [64], CAN [54], Tapestry [72], are not adequate for supporting flexibility requirement for content sharing P2P systems, which warrant minimum flexibility of partial keyword matching. This inadequacy stems from mainly for two reasons. Firstly, DHT-techniques use numeric distance based clustering of hashed keywords which is not suitable for partial keyword matching. Secondly, DHT-techniques cannot handle *common keywords problem* well. Popular keywords can incur heavy load on the peers responsible for these keywords; as a result, the distribution of load will become unbalanced among the participating peers.

Inability to support partial keyword matching is considered a handicap for DHT-techniques. In the last few years a number of research efforts have focused on extending DHT-techniques for supporting keyword search. Most of these approaches adopted either of the following two strategies:

- Build an additional layer on top of an existing DHT routing mechanism. The aim is to reduce the number of DHT lookups per search by mapping related keywords to nearby peers on the overlay. This strategy is proposed in a number of research works including [37, 44, 61, 67] .
- Combine structured and unstructured approaches in some hierarchical manner to gain the benefits of both paradigms. Few research works, including [28, 36, 66], focus on this strategy.

A *generic inverted index* [31] on top of a DHT-based routing can be used for achieving an expressiveness level of partial-keyword matching. In this approach, a keyword is translated into a routing key in two steps. First, the keyword is fragmented into η-grams. Then each η-gram is hashed and stored at the responsible peer on the DHT overlay. The hashed η-grams form an inverted index, where an advertised η-gram can be discovered by specifying its hash value. This approach will solve partial keyword matching problem in $O(\omega \log N)$ time, where ω is the number of η-grams in a query and N is the number of peers in the system, assuming that the underlying DHT network has logarithmic routing efficiency.

Keyword fusion [44] is another inverted indexing mechanism on top of Chord routing. Supported level of query expressiveness is keyword search only. A document advertised with keywords $\{k_1, k_2, \ldots, k_t\}$ is routed to peers responsible for keys $h(k_1), h(k_2), \ldots, h(k_t)$, where $h(\cdot)$ is the DHT hash function. To reduce the number of DHT-lookups per search, a system-wide dictionary of common keywords is maintained. A query is routed using the most specific keyword and then filtered using the more common keywords specified in the query. Thus the translation function filters out common keywords and then applies hashing. This strategy suffers from two problems. Firstly, the advertisement overhead is significant and proportional to the number of keywords. Secondly, maintaining the global dictionary for common keywords is not suitable for large, dynamic networks.

Joung et al. [37] proposed a distributed indexing scheme, build on a logical, d-dimensional hypercube vector space over Chord routing. In this scheme each advertisement is translated into a d-bit vector according to its keyword set (similar to Bloom filter construction). They treat d-bit vectors as points in d-dimensional hypercube. No restriction on the mapping of a d-dimensional point to a 1-dimensional key space (required for Chord) has been specified. An advertisement is registered to the peer responsible for the d-bit advertisement vector. A query vector (say Q) is computed in the same manner as the advertisement vector. A query is routed to all the peers in the Chord ring that are responsible for a key (say P_i) that is a superset of the query vector Q. Number of DHT lookups per search and query is significant for this approach.

The work by Joung et al. [37] and the inverted indexing method used in Keyword Fusion [44] represent the two extremes of advertisement and query traffic trade off. In [37], an advertisement is registered at one peer (responsible for the advertised key) and a query is routed to all possible peers that may contain a matching advertisement. On the other hand, in Keyword Fusion[44] an advertisement is registered at all the peers responsible for the advertised keywords and the query is routed to the peer responsible for the most uncommon keyword specified in the query.

pSearch [67] utilizes Information Retrieval (IR) techniques to construct the translation function on top of CAN routing for facilitating content-based full-text search. Keywords associated with an advertised document (or query) are represented as unit vectors. IR techniques like vector space model (VSM) and latent semantic indexing (LSI) are used to compute a unit vector from the keyword list specified in an advertisement (or a query). Similarity between a query and an advertisement (or between two advertisements) is measured using the dot product of the vector representation of the corresponding advertisement and query. Semantically close advertisements and queries are expected to be translated to geometrically close point vectors in the Cartesian space. Now the semantic point vectors from LSI or VSM are treated as geometric points in the Cartesian space of CAN. CAN partitions a d-dimensional, conceptual, Cartesian space into zones and assigns each zone to a peer. However this mapping technique (from LSI/VSM to d-dimensional CAN space) uses the same dimensionality for LSI space and CAN. Thus it needs to have a priori knowledge of the possible keywords (or terms) in the whole system. In reality there can be thousands of possible keywords, and CAN performance degrades at higher dimensions.

Squid [61] has been designed to support partial prefix matching and range queries on top of DHT-based structured P2P networks. It uses Hilbert Space-filling Curve (HSFC) [58] for translating keywords to keys on top of Chord routing mechanism. HSFC is a special type of locality preserving hash function that can map points from a d-dimensional grid (or space) to a 1-dimensional curve in such a way that the nearby points in d-dimensional space are usually mapped to adjacent values on the 1-dimensional curve. Squid converts keywords to base-26 (for alphabetic characters) numbers. A d-dimensional point is constructed from d keywords specified in the query or advertisement. Then a d-dimensional HSFC is used to translate a d-dimensional region (i.e., set of points) specified by the query into a set of curve segments in 1-dimension. Finally, each segment is searched using a Chord-lookup followed by a local flooding. Squid supports partial prefix matching (e.g., queries like compu* or net*) and multi-keyword queries; however, Squid does not have provision for supporting true inexact matching of queries like **net**. Another major problem is that the number of (partial) keywords specified in a query or advertisement is bounded by the dimensionality d of the HSFC in use.

MKey [36] is a hybrid approach to keyword search. Architecturally there exists a DHT (here Chord) backbone. A backbone node in the Chord ring works as a head for a cluster of nodes, organized in an unstructured fashion. Search within a cluster is based on flooding. On the other hand, Bloom filter is used as index in the backbone. But DHT techniques do not allow Hamming distance based indexing as required for matching Bloom filters. For allowing pattern matching on Chord, the following strategy is used. Nodes on the Chord ring are allowed to have an ID with at most two 1-bits. An advertisement pattern, say 01010111, is advertised to peers 01010000, 00000110 and 00000001; i.e., DHT-keys are obtained from an advertisement pattern by taking pairs of 1-bits in sequential order from left to right. To construct DHT-keys from a query pattern, say 01010011, only the leftmost three 1-bits are used. In this example the 1-bits at 2nd,4th and 7th positions. The DHT-keys are obtained by taking the 1-bit in center position (here 4th) and another bit within the left position (here 2nd) and the right position (here 7th). Hence for the query pattern 01010011, generated DHT-keys are 01010000, 00110000, 00011000, 00010100 and 00010010. Evidently the number of DHT-lookups per search or advertisement depends linearly on the number of keywords and the size of the used Bloom-filter. This can be more inefficient than a generic inverted indexing mechanism for inappropriate parameter settings. Besides, the nodes on Chord ring may become performance bottlenecks for the system.

There exists only a few non-DHT structured approaches to the search problem in P2P networks. *SkipNet* [32] and *SkipGraph* [9] are prominent among them. Both of these approaches use *Skip List* [52] for routing. A skip List is a probabilistic data structure consisting of a collection of ordered linked lists arranged into levels. The lowest level (i.e., level 0) is an ordinary, ordered linked list. The linked list in level i skips over some elements from the linked list at level $(i-1)$. An element in level i linked list can appear in level $(i+1)$ linked list with some predefined, fixed probability, say p. Storage overhead can be traded for search efficiency by varying p. Search for an element say Q starts at the topmost level. Level i list is sequentially searched until Q falls within the range specified by current element and next element in the list. Then the search recurs to level $i-1$ list from the current element until level 0 is reached. In both SkipGraph and SkipNet, nodes responsible for the upper level elements of the Skip List become potential hot spots and single points of failure. To avoid this phenomena, additional lists are maintained at each level.

5.2 Un-structured and Semi-structured Techniques

Unstructured systems identify objects by keywords. Advertisements and queries are expressed in terms of the keywords associated with the shared objects. Structured systems, on the other hand, identify objects by keys, generated by applying one-way hash function on keywords associated with an object. Key-based query routing is much efficient than keyword-based unstructured query routing. The downside of key-based query routing is the lack of support for partial-matching semantics as discussed in the previous section. Unstructured systems, utilizing blind search methods such as *Flooding* and *Random-walk*, can easily be modified to support partial-matching queries. But, due to the lack of proper routing information, the generated query routing traffic would be very high. Besides, there would be no guarantee on search completeness.

Many research activities are aimed at improving the routing performance of unstructured P2P systems. Different routing hints are used in different approaches. In GIA [19], routing is biased by peer capacity; queries are routed to peers of higher capacity with higher probability. In APS [68, 71], peers learn from the results of previous routing decisions and bias future query routing based on this knowledge. In Associative Search [22], peers are organized based on common interest, and restricted flooding is performed in different interest groups. Many research works (GIA [19], NSS [42, 71], *etc.*) propose storing index information from peers within a radius of 2 or 3 hops on the overlay network. All of these techniques reduce the volume of search traffic to some extent, but none provides guarantee on search completeness.

Bloom filters are used by a few unstructured P2P systems as translation function for improving query routing performance. In NSS [42] each peer stores Bloom filters from peers one or two hops away. Experimental results presented in this work show that logical OR-based aggregation of Bloom filters is not suitable for indexing information from peers more than one hop away. In PLR [55] each peer stores a list of Bloom filters, named Attenuated Bloom filter, per neighbor. The *i*th Bloom filter in the list of Bloom filters for neighbor M summarizes the resources that are $i-1$ hops away from neighbor M. A query is forwarded to the neighbor with a matching Bloom filter at the smallest hop-distance. This approach aims at finding the closest replica of a document with a high probability.

5.3 Summary

Table 1 summarizes the query semantics, translation functions and routing mechanisms as observed in different search techniques in P2P content sharing domain as discussed in this section.

Table 1 Components of selected search techniques in P2P content sharing

P2P content sharing					
References	Name	Query	Translation	Type	Routing Mechanism
[44]	Keyword fusion	Multi-keyword	Inverted index	AR	Chord
[37]	Joung et al.	Multi-keyword	Query superset	AR	Chord
[67]	pSearch	Full text, multi-keyword	VSM/LSI	AR	CAN
[61]	Squid	Prefix match	Hilbert SFC	AR	Chord
[36]	MKey	Subset match	Query superset	AR	Chord +Flooding
[32]	SkipNet	Prefix match	Flat	AR	Skip List
[19]	GIA	Partial keyword	Flat	CR	Capacity bias
[68]	APS	Partial keyword	Flat	CR	Result bias
[42]	NSS	Multi-keyword	Bloom filter (BF)	SR	Controlled flood
[55]	PLR	Multi-keyword	Attenuated BF	SR	Hint bias

6 Search Techniques in P2P Service Discovery

Many service discovery systems rely on a three-party architecture, composed of clients, services and directory entities. Directory entities gather advertisements from service providers and resolve queries from clients. Major protocols for service discovery from industry, like SLP [30], Jini [65], UPnP [47], Salutation [59], etc., assume a few directory agents, and do not provide any efficient mechanism for locating Service Descriptions. Solutions from academia, like Secure Service Discovery Service (SSDS) [24] and Twine [11], target Internet-scale service discovery and face the challenge of achieving efficiency and scalability in locating Service Descriptions based on partial information.

Secure Service Discovery Service (*SSDS*) [24] arranges directory entities in a tree-like structure and uses hierarchy routing. It uses Bloom filters for translating service descriptions into routing signatures. Bit-wise OR-base aggregation scheme is adopted for reducing the volume of index information at higher level directory entities in directory tree. In SSDS an advertisement can be discovered by specifying a subset of the advertised property-value list in the query expression. SSDS suffers from load-balancing problem and is vulnerable to the failure of higher level directory entities along the directory tree.

Twine [11] uses a hierarchical naming scheme and relies on Chord as the underlying routing mechanism. A resource is described using a *name-tree*, composed of the properties and values associated with the resource. Hierarchical relations between properties are reflected in the tree, e.g., while describing the location of a resource, “room no.” appears as a child of the “building” in which it resides. The translation function in Twine generates a set of strands (substrings) from the advertisement or query (which are expressed in XML format), computes keys for each of these strands, and finally uses these keys for the search or advertisement process. The stranding algorithm in Twine is designed to support partial prefix matching within a name-tree. The number of DHT-lookups increases with the number of property-value pairs in the advertisement (or query) and consequently the amount of generated traffic becomes high. Load-balancing is another major problem in this system. Peers responsible for small or popular strands become overloaded, and the overall performance degrades.

Web Services (WS) [14] provide a standard way of interoperating between different software applications, running on a variety of platforms and/or frameworks. Universal Description, Discovery and Integration (UDDI) [69] is the defacto standard for WS discovery. Many research activities are devoted to enhancing and overriding the legacy UDDI specification thriving for efficiency, scalability and flexibility in the discovery mechanism. A detailed survey of such activities can be found in [29]. Table 2 summarizes some of the proposed architectures for WS discovery. Based on the use of WS ontologies, these approaches can be broadly classified as *semantic-laden* and *semantic-free*. Semantic-laden approaches rely on WS ontology mapping techniques like OWL (Web ontology language) [8] or DAML (DARPA Agent Markup Language) [16] for incorporating intelligence to the discovery process, i.e., for intelligently mapping conceptually related terms in queries and advertisements. Semantic-free approaches, on the other hand, do not utilize WS ontology mapping techniques. These approaches are closely related to the traditional service discovery systems. A number of research work in this category rely on locality preserving hash techniques for translating queries to semantically close advertisements.

6.1 Summary

Table 3 summarizes the query semantics, translation functions and routing mechanisms as observed in different search techniques in service discovery domain as discussed in this section.

Table 2 Summary of Web service discovery architectures

Centralized	Registry	Authoritative, centrally controlled store of service descriptions, e.g., UDDI registry [69]	
	Index	Non-authoritative, centralized repository of references to service providers; see [14] for details. Web crawlers are used for populating an index database	
Decentralized	Federation	Publicly available UDDI nodes collaborate to form a federation and act together as a large scale virtual UDDI registry [56]	
	P2P-based	Semantic-laden	In [60] peers are arranged into a hypercube topology [25] and ontology [70] is used to facilitate efficient and semantically-enabled discovery. An agent-based approach is proposed in [48]. It uses DAML [16] representation for ontology and relies on unstructured search techniques.
		Semantic-free	Both [43, 62] use Chord overlay for indexing and locating service information. Reference [43] extracts property-value pairs from service descriptions and uses MD5 hashing. Reference [62] uses Hilbert Space Filling Curves for mapping similar Service Descriptions to nearby nodes in the Chord ring. These two approaches are similar to Twine [11] and Squid [61], respectively. In [35], another Chord based solution has been proposed. Here, the ID-space is partitioned in numerically ordered subspaces, and each peer in the Chord-ring maintains links to one peer in each subspace in addition to the regular Chord links.

Table 3 Components of selected search techniques in service discovery

Service discovery					
References	Name	Query	Translation	Type	Routing
[30]	SLP	LDAP filter	Flat	CR	Flooding
[24]	SSDS	Subset/PV-list	Bloom filter	SR	Global hierarchy
[11]	Twine	Subtree match	Stranding + hash	CR	Chord
[43]	PWSD	XML path prefix	Stranding + hash	CR	Chord
[62]	Schmidt et al.	Prefix match	Hilbert SFC	CR	Chord
[60]	Schlosser et al.	Semantic match	Ontology concept $\rightarrow$ d-coord.	CR+ AR	2-tier hypercube

7 Search Techniques in Distributed XML Databases

Several research works on distributed XML databases have adopted DHT techniques, such as Chord [64], CAN [54] and Hypercube [60], for routing. A number of these proposals, including [13, 17, 27], rely on Chord as the underlying P2P substrate. Hypercube topology has been used in [49].

XP2P [13], uses XML data model for schema representation, and provides support for resolving XPath [38] queries. Any XML document can be represented as a tree, and an XPath query is used to specify a subtree using a prefix-path originating from the root of the document. For supporting partial prefix-path matching, all possible paths, originating from the root, have to be registered with the Chord ring. To reduce the number of paths to be hashed in the Chord ring during the advertisement and query process, XP2P adopts a fingerprint construction technique presented in [53]. In this technique, the fingerprint of a binary string $A(t) = (a_1, a_2, \dots, a_m) = a_1 \times t^{m-1} + a_2 \times t^{m-2} + \dots + a_m$ is computed as $f(A) = A(t)\%P(t)$, where $P(t)$ is an irreducible polynomial. A useful property of the fingerprint function, utilized by XP2P, is that $f(A \odot B) = f(f(A) \odot B)$, where $\odot$ is the concatenation operator.

Galanis et al. [27] presented a framework for supporting XPath queries on top of Chord routing. XPath queries of the form $/a_1[b_1]/a_2[b_2]/\dots/a_n\, op\ value$ and queries containing relative path operator (i.e., $//$) are supported. Here, a_i is an element in an XML document, b_i is an XPath expression relative to element a_i, op is an XPath operator like $=$ or $<$, and *value* is an atomic element in the XML document. The core idea is to build a distributed catalog, where a peer in the Chord ring stores all the prefix-paths for a given element in any XML document stored in the network. In other words, if E is an element in some XML files, then the peer responsible for the key $hash(E)$ stores all the absolute paths (i.e., $/a_1/a_2/\dots/E$) leading to E in any document stored in the network and the contact information of the peers storing those documents. An XPath query of the form $/a_1/a_2/\dots/a_k//E$ is routed to the peer (say N) responsible for the key $hash(E)$ and the list of all peers containing XML documents matching the query are extracted. Finally the query is forwarded and executed in the corresponding peers.

RDFPeers [17] uses Resource Description Framework (RDF) [41] for document representation and Chord for routing. An RDF document is contains many $< Resource, Property, Value >$ triples presented in XML format. A triple, say $< R, P, V >$, is stored in three peers (in the Chord ring) responsible for the keys $hash(R)$, $hash(P)$ and $hash(V)$, respectively. For string literals SHA1 hash function is used. For numeric values (in the *value* component of a RDF-triple) locality preserving hash function is used. A query can be constructed by specifying any of the three components in a triple. In RDFPeers each document has to be indexed at three peers, which results into increased increased advertisement and update traffic.

PeerDB [50] uses an agent-based framework on top of unstructured P2P overlay to achieve distributed data sharing. To accommodate heterogeneity in schema definitions from autonomous peers in the system, PeerDB associates keywords as synonyms with each schema and elements under that schema. These keywords are used as a means of semantic mapping and for finding semantically similar schemas using P2P keyword search techniques. Mobile agents are sent to appropriate peers and a query is executed locally at the target peer, which helps in reducing the volume of network traffic.

A hybrid technique, named *Humboldt discoverer*, has been presented in [33]. RDF [41] has been used for describing an advertised resource. SPARQL (Simple Protocol and RDF Query language) [51] has been used for constructing query expressions. SPARQL is a query language for RDF documents that allows formation of complex queries involving relational and logical operators. Routing is done using a three tier architecture, where peers are classified as bottom, middle or top tier peers. Bottom tier peers provide information sources. These peers are clustered into many groups based on the similarity of used ontologies. A middle tier peer is responsible for an ontology and manages a single cluster of bottom tier peers. Middle tier peers advertise their existence to top tier peers, which are organized in a Chord ring and are addressed by the hash of the URIs of the ontologies. In effect, middle tier peers covering the same ontology are grouped under the same top level peer. To resolve a query, all the required ontologies are first determined. For a given ontology, the set of responsible middle tier peers can be reached through the top tier Chord network. Finally, the query is forwarded to each of the middle-tier peers that are responsible for the ontology.

7.1 Summary

Table 4 summarizes the query semantics, translation functions and routing mechanisms as observed in different search techniques in distributed XML database domain as discussed in this section.

Table 4 Components of selected search techniques in PDBS

P2P databases					
Ref	Name	Query	Translation	Type	Routing
[13]	XP2P	XPath(absolute)	Fingerprint	AR	Chord
[27]	Galanis et al.	XPath(relative)	XML element hash	AR	Chord
[17]	RDFPeers	Partial RDF triple	RDF element hash	AR	Chord
[50]	PeerDB	SQL	Synonym/flat	CR	Flooding
[33]	Humboldt Discoverer	SPARQL/RDF	URI-hash+Flat	AR+CR	Chord+ Controlled flooding

8 The DPM Abstraction

It is evident from the foregoing survey that flexibility in query expressiveness is essential for a search mechanism in all three domains, considered so far. This section presents an abstraction for the search problems in different application domains into a generic framework or problem formulation, called Distributed Pattern Matching (DPM).

Among the requirements of a search mechanism discussed in Section 3, flexibility deals with query language semantic and the rest relate to system performance and thus are implementation specific. The DPM construct encapsulates the flexibility requirement, and any solution to the DPM problem should focus on the performance specific requirements.

8.1 Distributed Pattern Matching (DPM)

The DPM problem can be considered as a distributed version of the traditional pattern matching problem or more specifically the subset matching problem. The generic pattern matching problem and its variants have been extensively studied in Computer Science literature. Given a *text* (or raw data) $\mathbb{P}$ and a *pattern* $\mathbb{Q}$, the generic problem of pattern matching is to locate (all) the *occurrence*s of $\mathbb{Q}$ in $\mathbb{P}$. The definition of *text*, *pattern* and *occurrence* depends on the application domain. The *text* and *pattern* are two dimensional arrays in Image Processing applications, *string*s in Text Editing systems, trees in *tree pattern matching* [40], and arrays of sets in *subset matching* [23]. Variations in the definition of *occurrence* include exact matching, parameterized matching [10], approximate matching and matching with "don't cares" [6].

The variant of the pattern matching problem considered in this work is closely related to the *subset matching* problem. In subset matching, pattern $\mathbb{Q} = \{Q_1, Q_2 \ldots Q_m\}$ and text $\mathbb{P} = \{P_1, P_2 \ldots P_n\}$ are collections of sets of characters drawn from some alphabet $\sum$. A pattern $\mathbb{Q}$ occurs at text position i if the set Q_j is a subset of the set P_{i+j-1}, for all $1 \leq j \leq m$.

The *Distributed Pattern Matching (DPM)* problem is defined as a variant of the *subset matching* problem with the following restrictions:

- The pattern $\mathbb{Q}$ has a single element in its array, i.e., $m = 1$ and $\mathbb{Q} = Q$.
- The n elements of $\mathbb{P}$ are distributed across a large number of networked nodes.

In many cases, bit-vectors of length $|\sum|$ are used to present texts (i.e., P_i) and pattern (i.e., Q), where a 1 (or 0) at the ith bit of a bit vector resembles the presence (or absence) of the ith symbol in $\sum$. In the DPM formulation it is assumed that each element of $\mathbb{P}$ (i.e., P_i) summarizes the identifying properties (e.g., keywords, service description) of a shared object (e.g., a file or a service). One possible form of such a pattern is a Bloom filter [12] obtained by hashing the properties associated with a shared object.

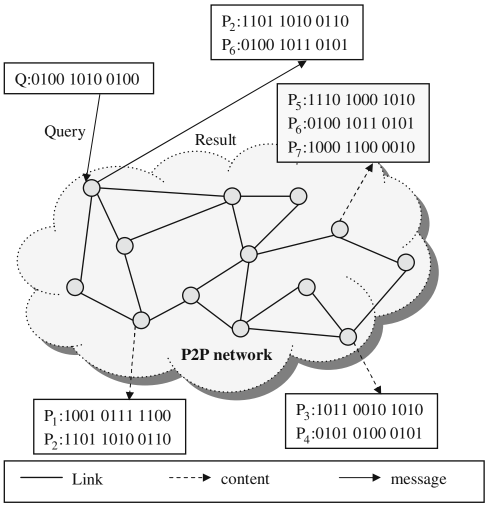


Fig. 9 The distributed pattern matching (DPM) problem. ($|\sum| = 12$)

Figure 9 presents a pictorial view of the DPM problem. In this figure, P_i represents an advertised pattern and Q stands for a search pattern. The properties of shared objects are encoded in the advertisement pattern, $P_1 - P_7$ in Fig. 9. On the other hand, a query (Q in Fig. 9) is constructed by encoding the desired properties in a bit vector. The encoding technique must ensure that if the desired properties specified in a query is a subset of the properties of an shared object then the 1-bits of the corresponding query pattern must be a subset of the 1-bits in the advertised pattern. In other words, the result of a search should contain all the advertised patterns (P_2 and P_6 for the example in Fig. 9) that are supersets of the search pattern (Q in Fig. 9).

8.2 Mapping to DPM Framework

This section describes possible ways of encoding advertisements and queries for the three application domains.

8.2.1 P2P Content Sharing

An advertisement in a P2P content sharing system consists of a number of keywords describing the content being shared. For a file-sharing P2P system, it is unusual for a user to know the exact name of an advertised file. Instead, queries are based on a subset of the (partial) keywords that may be present in the advertisement. Bloom filters can be used for encoding advertised and queried keywords in the following manner. An advertisement Bloom filter can be constructed using the trigrams extracted from the keywords associated with an advertised document. Similarly, a query Bloom filter can be computed from the keywords presented in the query string. Thus there will be one Bloom filter per advertisement or query. Subset relationship between advertised and queried trigrams will hold for advertisement and query Bloom filters. For example, in Fig. 10 trigrams for the first query constitute a subset of the advertised trigrams; as a result query pattern Q_1 is a subset of the advertisement pattern P. On the other hand, trigrams from the second query do not correspond to any subset of the advertised trigrams, and with high probability Q_2 will not be a subset of P.

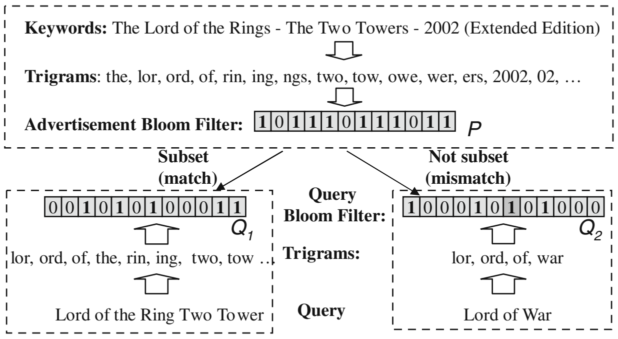

Keywords: The Lord of the Rings - The Two Towers - 2002 (Extended Edition)
Trigrams: the, lor, ord, of, rin, ing, ngs, two, tow, owe, wer, ers, 2002, 02, …
Advertisement Bloom Filter: 1 0 1 1 1 0 1 1 1 0 1 1
P
Subset (match)
Not subset (mismatch)
0 0 1 0 1 0 1 0 0 0 1 1
Q1
Query Bloom Filter:
1 0 0 0 1 0 1 0 1 0 0 0
Q2
lor, ord, of, the, rin, ing, two, tow ...
Trigrams:
lor, ord, of, war
Lord of the Ring Two Tower
Query
Lord of War

Fig. 10 Partial keyword matching using DPM

8.2.2 Service Discovery

For most service discovery systems a service description is advertised as a set of property-value pairs and a query for a service consists of a subset of the advertised property-value pairs. All of the decentralized techniques for (Web) Service discovery are aimed at achieving flexibility (i.e., partial matching capability) without sacrificing efficiency in the search mechanism. The basic problem of service description matching and semantic matching in a distributed environment can be mapped to the DPM problem in different ways. A few possibilities are listed below.

- The simplest way of mapping a service description to a Bloom filter is to treat each property-value pair as set elements, see Fig. 11. Query pattern can be constructed in a similar fashion.
- While generating an advertisement pattern from a Service Description, multiple synonyms of the properties and values can be hashed and inserted into the Bloom-filter. This will require larger Bloom-filters, yet will enable one to discover a service by specifying any of the synonyms of a property (or value) specified in the advertisement.
- It is also possible to use ontologies during Bloom filter construction, rather than using simple synonym list. Use of Ontology can be more efficient than synonym list if there exists a global Ontology in the system.

8.2.3 Distributed XML Databases

Distributed search in distributed XML databases faces two new challenges, in addition to the ones present in P2P content sharing and service discovery: continuously changing schemas in the system and the requirement for semantic mapping. It is possible to cope with these challenges using the DPM construct.

For P2P database systems, as shown in Fig. 11, XML documents are used as advertisements and XPath [38] is the most commonly used query language. Figure 11 presents an XPath query of the form $/a_1[b_1]/a_2[b_2]\dots/a_n[b_n]$. Here, a_i is an element in an XML document, b_i is an XPath expression relative to element a_i. In this case, path prefixes from an XML document or the XPath expression (e.g., $/a_1$, $/a_1/b_1$, $/a_1/a_2\dots$) can be used as the *set element*s for Bloom filter construction.

In order to accommodate continuously changing schemas, advertised patterns should be constructed from both the descriptive properties and values of a shared object. Thus, schema information gets incorporated within each advertisement. The requirement for semantic mapping can be satisfied in few ways, including the ones discussed in the previous section. It is also possible to reserve a pre-specified number of bits in the advertisement/query pattern, and use these bits as a separate Bloom-filter for storing the ontologies used by the advertisement/query. Query routing mechanism can use this additional information for making semantic laden decisions at each hop.

8.3 Known Solutions to the DPM Problem

An efficient solution to the DPM problem is expected to satisfactory solve the search problem in the three important application domains discussed so far. In this section two solutions to the DPM problem, namely DPMS [3] and Plexus [4], are presented.

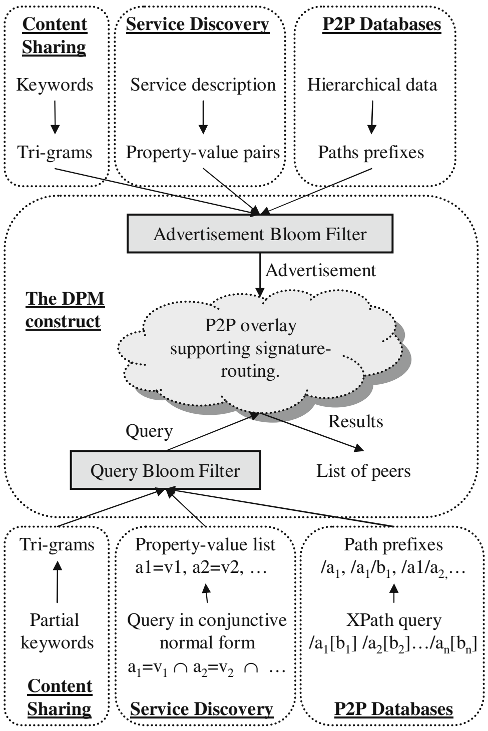


Fig. 11 Mapping different problems to DPM framework

8.3.1 DPMS: Distributed Pattern Matching System

In DPMS a peer can act as a *leaf peer* or *indexing peer*. Leaf peers reside at the bottom level of the indexing hierarchy and act as the document source for the system. An indexing peer, on the other hand, stores indices from other peers (leaf peers or indexing peers). A peer can join different levels of the indexing hierarchy and can simultaneously act in both the roles. Indexing peers get arranged into a lattice-like hierarchy for indexing and disseminating advertised patterns from the leaf peers using repeated aggregation and replication.

The structure of the indexing hierarchy and the amount of replication are controlled by two system-wide parameters, namely replication factor R and branching factor B. Patterns advertised by a leaf peer are propagated to R^l indexing peers at level l. On the other hand, an indexing peer at level l contains patterns from B^l leaf peers. Due to repeated (lossy) aggregation, information content of the aggregates reduces while climbing up the indexing hierarchy.

Indexing peers at level l arrange into R^l groups, numbered from 0 to $(R^l - 1)$ (see Fig. 12). In the ideal case, all the indexing peers in a single group (at any level) collectively cover all the leaf peers in the system. A peer at level l and group

g $(0 \leq g < R^l)$ is responsible for transmitting its aggregated information to R parents at level $(l+1)$. Each parent belongs to a different group in range $[g \times R, (g+1) \times R)$, respectively.

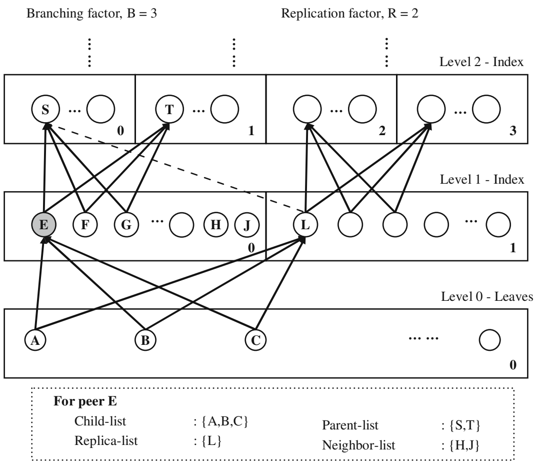


Fig. 12 Index distribution architecture. All the peers interacting with peer *E* are labelled. Group number is printed at the bottom right corner of each box

Peers at level l and group g organize into subgroups (referred to as siblings) of size B to forward their aggregated information to the same set of parents. Thus each group in range $[g \times R, (g+1) \times R)$ at level $(l+1)$ will contain a peer replicating the same index information. This provides redundant paths for query routing and increases tolerance to peer failure.

A don't care based aggregation is used in DPMS. While aggregating to patterns (or aggregate) a don't care ("X") is inserted in the bit-positions at which the constituent patterns disagree. The resulting aggregates retain parts from the constituent patterns or aggregates. A 1-bit (or 0-bit) in such an aggregate indicate that all of the patterns contributing to this aggregate had 1 (or 0) at the corresponding position. However incorporating this extra information (i.e., X's) incur some space overhead, which can be minimized by compressing the aggregates using huffman coding or run length encoding during transmission through the network.

A query is executed in three phases: ascending phase, blind search phase and descending phase. In *ascending phase* the query message climbs the indexing hierarchy until a match is found or it reaches the highest level. If no match is found int he ascending phase the query enters *blind search phase*, where the query message is flooded in one of the groups at the topmost level. If any match exists then it will be discovered in this phase. Finally, in *descending phase* the query message is forwarded to the target leaf peer(s) using the aggregation trail along the indexing hierarchy.

8.3.2 Plexus

Plexus has a *partially decentralized* architecture involving superpeers. It adopts a *structured routing* mechanism derived from the theory of *Linear Covering Codes*.[1] Indexing and routing in Plexus is based on the Hamming distance between the advertised and queried patterns, in contrast to the numeric distance based routing adopted in traditional DHT-approaches. This property makes subset matching capability intrinsic to the underlying routing mechanism. Plexus achieves better resilience to peer failure by utilizing replication and redundant routing paths. Routing efficiency in Plexus scales logarithmically with the number of superpeers.

In Plexus, advertisements and queries are routed to two different sets of peers in such a way that the queried set of peers and the advertised set of peers have at least one peer in common, whenever a query pattern constitute a subset of the 1-bits, as present in an advertised pattern. As explained in Fig. 13, a linear covering code partitions the entire pattern space $\mathbb{F}_2^n$ into Hamming spheres. The codeword at the center of each Hamming sphere is selected as unique representative for that cluster. Now the basic concept is to map a query pattern Q to a set of codewords ($\mathscr{Q}(Q) \subset \mathscr{C}$) and to map an advertised pattern P to another set of codewords ($\mathscr{A}(P) \subset \mathscr{C}$), such that $\mathscr{Q}(Q)$ and $\mathscr{A}(P)$ has *at least one* codeword in common whenever the 1-bits of Q constitute a subset of the 1-bits in P. Mathematically,

$$Q \subseteq P \implies \mathscr{Q}(Q) \cap \mathscr{A}(P) \neq \emptyset \tag{1}$$

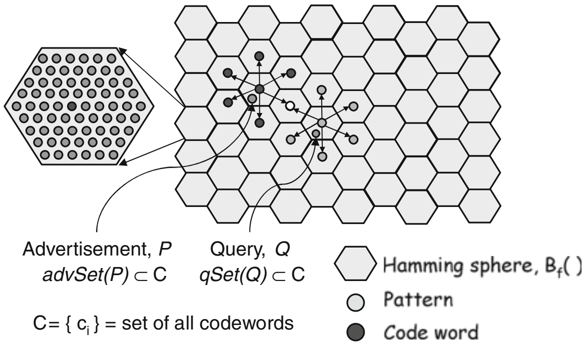
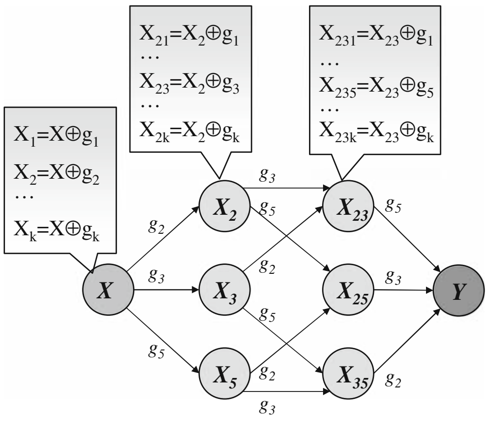


Fig. 13 Hamming distance based indexing in plexus

Any codeword in a linear covering code can be generated by using bit-wise XOR operation of any combination of specially chosen k codewords, labelled

[1] A linear covering code $(n,k,d)f$, over a linear space $\mathbb{F}_2^n$, is a *subspace* $\mathscr{C} \subset \mathbb{F}_2^n$. Each element in $\mathscr{C}$ is called a *codeword*. Here, $|\mathscr{C}| = 2^k$ and d is the minimum Hamming distance between any two codewords. The *covering radius* f is the smallest integer such that every vector $P \in \mathbb{F}_2^n$ is covered by at least one $B_f(c_i)$. Here, $B_f(c_i) = \{P \in \mathbb{F}_2^n | d(P, c_i) \leq f\}$ is the *Hamming sphere* of radius f centered at codeword c_i.

$G = [g_1, g_2, \dots g_k]$. In other words, any codeword Y can be generated from any other codeword X as follows: $Y = (X \oplus g_{i_1} \oplus g_{i_2} \oplus \dots \oplus g_{i_t})$, where $g_{i_1}, g_{i_2}, \dots g_{i_t} \in G$ and $\oplus$ is bitwise XOR operation. For a simple implementation each superpeer in the network is assigned a codeword. A superpeer with codeword X links to k other superpeers with codeword $X_i = X \oplus g_i$, $(1 \le i \le k)$. The routing process in Plexus can be best explained by the example in Fig. 14, which shows the possible routes from the superpeer with codeword X to the superpeer with codeword $Y = g_2 \oplus g_3 \oplus g_3$. Superpeer X will forward the message to any of $X_2 (= X \oplus g_2)$, $X_3 (= X \oplus g_3)$ or $X_5 (= X \oplus g_5)$, who are one hop nearer to Y than X. If the message is forwarded to say X_2 then X_2 can route the message to Y via $X_{23} (= X \oplus g_2 \oplus g_3)$ or $X_{25} (= X \oplus g_2 \oplus g_5)$. Number of hops required for routing in this mechanism is logarithmic on the number of superpeers in the network.



$$Y = X \oplus g_2 \oplus g_3 \oplus g_5$$

Fig. 14 Possible paths of routing from peer X to peer Y

8.3.3 Summary

Table 5 summarizes the query semantics, translation functions and routing mechanisms as observed in DPMS and Plexus.

9 Conclusion

Large scale distributed systems including P2P content sharing, service discovery and distributed XML data require flexible, efficient and robust search mechanism due to the volatility in node population and dynamism in advertised objects. Based

Table 5 Components of selected search techniques in service discovery

Distributed pattern matching				
References	Name	Query	Translation	Routing
[3]	DPMS	Subset matching	Feature extraction+Bloom filter	Hierarchical +selective flooding
[4]	Plexus	Subset matching	Feature extraction+Bloom filter	Structured – Linear code based

on routing mechanism contemporary search techniques can be broadly classified as content routing, address routing and signature routing. Content routing techniques preserves query semantics and thus provide better search flexibility; but routing traffic is high for these techniques. Address routing techniques, on the other hand are efficient in routing traffic but offered flexibility in query expressiveness is inadequate. Signature routing can be a good candidate for balancing the trade off between efficiency and flexibility requirements.

The Distributed Pattern Matching (DPM) framework provides an abstract formulation for pattern matching in large scale distributed systems. Keyword search for content-sharing P2P systems, partial Service Descriptions matching for service discovery systems and semantic laden data retrieval for for distributed XML databases can be mapped to the DPM abstraction. DPMS and Plexus are two solutions for the DPM problem. DPMS is a hierarchical signature routing technique, while Plexus is a hamming distance based address routing technique. Routing efficiency in both DPMS and Plexus scales logarithmic on network size. These two solutions as well as future solution to the DPM problem can be tuned to resolve the search problem in the three application domains discussed in this chapter.

References

1. The Gnutella website, http://www.gnutella.com
2. Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., Lilley, J.: The Design and Implementation of an Intentional Naming System. In: Symposium on Operating Systems Principles, pp. 186–201 (1999)
3. Ahmed, R., Boutaba, R.: Distributed pattern matching: A key to flexible and efficient P2P search. IEEE Journal on Selected Areas in Communications (JSAC) **25**(1), 73–83 (2007)
4. Ahmed, R., Boutaba, R.: Plexus: A scalable Peer-to-Peer protocol enabling efficient subset search (2009). IEEE/ACM Transaction on Networking (TON)
5. Ahmed, R., Limam, N., Xiao, J., Iraqi, Y., Boutaba, R.: Resource and service discovery in large-scale multi-domain networks. IEEE Communications Surveys & Tutorials **9**(4) (2007)
6. Amir, A., Porat, E., Lewenstein, M.: Approximate subset matching with don't cares. In: Proc. of Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 305–306 (2001)
7. Androutsellis-Theotokis, S., Spinellis, D.: A survey of Peer-to-Peer content distribution technologies. ACM Computing Surveys **45**(2), 195–205 (2004)

8. Antoniou, G., van Harmelen, F.: Web Ontology Language: OWL. Handbook on Ontologies in Information Systems pp. 76–92 (2003)
9. Aspnes, J., Shah, G.: Skip graphs. In: Proc. of Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 384–393 (2003)
10. Baker, B.S.: A theory of parameterized pattern matching: algorithms and applications. In: Proc. of ACM Symposium on Theory of Computing (STOC), pp. 71–80 (1993)
11. Balazinska, M., Balakrishnan, H., Karger, D.: INS/Twine: A scalable Peer-to-Peer architecture for intentional resource discovery. In: Proc. of International Conference on Pervasive Computing, pp. 195–210. Springer-Verlag (2002)
12. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of ACM **13**(7), 422–426 (1970)
13. Bonifati, A., Matrangolo, U., Cuzzocrea, A., Jain, M.: XPath lookup queries in P2P networks. In: Proc. of the ACM international workshop on Web information and data management (WIDM), pp. 48–55. ACM Press, New York, NY, USA (2004)
14. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.:Web Service Architecture (2004). URL `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`
15. Brookshier, D., Govoni, D., Krishnan, N.: JXTA: Java P2P Programming. SAMS (2002)
16. Burstein, M.H., Hobbs, J.R., Lassila, O., Martin, D., McDermott, D.V., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T.R., Sycara, K.P.: DAML-S: Web Service description for the semantic web. In: Proc. of International Semantic Web Conference on The Semantic Web (ISWC), pp. 348–363. Springer-Verlag, London, UK (2002)
17. Cai, M., Frank, M.: RDFPeers: a scalable distributed RDF repository based on a structured Peer-to-Peer network. In: International World Wide Web Conference (WWW) (2004)
18. Chamberlin, D., Siméon, J., Boag, S., Florescu, D., Fernández, M.F., Robie, J.: XQuery 1.0: An XML query language. W3C recommendation, W3C (2007). `http://www.w3.org/TR/2007/REC-xquery-20070123/`
19. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making Gnutella-like P2P systems scalable. In: Proc. of ACM SIGCOMM, pp. 407–418 (2003)
20. Choon-Hoong, D., Nutanong, S., Buyya, R.: Peer-to-Peer Computing: Evolution of a Disruptive Technology, chap. 2–Peer-to-Peer Networks for Content Sharing, pp. 28–65. Idea Group Inc. (2005)
21. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. Lecture Notes in Computer Science (LNCS) **2009**, 46–66 (2001)
22. Cohen, E., Fiat, A., Kaplan, H.: Associative search in Peer-to-Peer networks: Harnessing latent semantics. In: Proc. of IEEE INFOCOM (2003)
23. Cole, R., Harihan, R.: Tree pattern matching and subset matching in randomized $o(n\log^3 m)$ time. In: Proc. of ACM Symposium on Theory of Computing (STOC), pp. 66–75 (1997)
24. Czerwinski, S.E., Zhao, B.Y., Hodes, T.D., Joseph, A.D., Katz, R.H.: An Architecture for a Secure Service Discovery Service. In: Proc. of International Conference on Mobile Computing and Networking (MOBICOM), pp. 24–35 (1999)
25. Decker, S., Schlosser, M., Sintek, M., Nejdl, W.: Hypercup – hypercubes, ontologies and efficient search on P2P networks. In: International Workshop on Agents and Peer-to-Peer Computing (2002)
26. Fuchs, M., Wadler, P., Robie, J., Brown, A.: XML schema: Formal description. W3C working draft, W3C (2001). Http://www.w3.org/TR/2001/WD-xmlschema-formal-20010925/
27. Galanis, L., Wang, Y., Jeffery, S., DeWitt., D.: Locating data sources in large distributed systems. In: Proc. of the VLDB Conference, (2003)
28. Ganesan, P., Sun, Q., Garcia-Molina, H.: Adlib: A self-tuning index for dynamic Peer-to-Peer systems. In: Proc. of the International Conference on Data Engineering (ICDE), pp. 256–257. IEEE Computer Society, Los Alamitos, CA, USA (2005)

29. Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A.: Web service discovery mechanisms: Looking for a needle in a haystack? In: International Workshop on Web Engineering (2004)
30. Guttman, E., Perkins, C., Veizades, J., Day, M.: Service Location Protocol (SLP), version 2. Tech. rep., IETF, RFC2608, http://www.ietf.org/rfc/rfc2608.txt (1999)
31. Harren, M., Hellerstein, J.M., Huebsch, R., Loo, B.T., Shenker, S., Stoica, I.: Complex queries in DHT-based Peer-to-Peer networks. In: Proc. of International Workshop on Peer-to-Peer Systems (IPTPS), pp. 242–259 (2002)
32. Harvey, N., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: SkipNet: A scalable overlay network with practical locality properties. In: Proc. of the USENIX Symposium on Internet Technologies and Systems (USITS) (2003)
33. Herschel, S., Heese, R.: Humboldt Discoverer: A semantic P2P index for PDMS. In: Proc. of the International Workshop Data Integration and the Semantic Web (DISWeb'05) (2005)
34. Howes, T.: The String Representation of LDAP Search Filters. USA (1997). RFC Editor
35. Hu, H., Seneviratne, A.: Autonomic Peer-to-Peer service directory. IEICE/IEEE Joint Special Section on Autonomous Decentralized Systems **E88-D**(12), 2630–2639 (2005)
36. Jin, X., Yiu, W.P.K., Chan, S.H.: Supporting multiple-keyword search in a hybrid structured Peer-to-Peer network. In: Proc. of IEEE International Conference on Communications (ICC), pp. 42–47. Istanbul (2006)
37. Joung, Y., Yang, L., Fang, C.: Keyword search in DHT-based Peer-to-Peer networks. IEEE Journal on Selected Areas in Communications (JSAC) **25**(1), 46–61 (2007)
38. Kay, M., Fernández, M.F., Boag, S., Chamberlin, D., Berglund, A., Siméon, J., Robie, J.: XML path language (XPath) 2.0. W3C recommendation, W3C (2007). Http://www.w3.org/TR/2007/REC-xpath20-20070123/
39. Koloniari, G., Pitoura, E.: Peer-to-Peer management of XML data: issues and research challenges. ACM SIGMOD Record **34**(2), 6–17 (2005)
40. Kosaraju, S.R.: Efficient tree pattern matching. In: Proc. of the 30th IEEE Symposium on the Foundations of Computer Science (FOCS), pp. 178–183 (1989)
41. Lassila, O., Swick, R.R.: Resource description framework (RDF) model and syntax specification. supersed work, W3C (1999). Http://www.w3.org/TR/1999/REC-rdf-syntax-19990222
42. Li, M., Lee, W., Sivasubramaniam, A.: Neighborhood signatures for searching P2P networks. In: Proc. of Seventh International Database Engineering and Applications Symposium (IDEAS), pp. 149–159 (2003)
43. Li, Y., Zou, F., Wu, Z., Ma, F.: PWSD: A scalable web service discovery architecture based on Peer-to-Peer overlay network. In: Proc. APWeb, Lecture Notes Ccomputer Science (LNCS), vol. 3007 (2004)
44. Liu, L., Ryu, K.D., Lee, K.: Supporting efficient keyword-based file search in Peer-to-Peer file sharing systems. In: Proc. of GLOBECOM (2004)
45. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured Peer-to-Peer networks. In: Proc. of the International Conference on Supercomputing (ICS), pp. 84–95 (2002)
46. Maymounkov, P., Mazireres, D.: Kademlia: A Peer-to-Peer information system based on the XOR metric. In: Proc. of International Workshop on Peer-to-Peer Systems (IPTPS), pp. 53–65. Springer-Verlag (2002)
47. Miller, B.A., Nixon, T., Tai, C., Wood, M.D.: Home networking with Universal Plug and Play. IEEE Communications Magazine pp. 104–109 (2001)
48. Montebello, M., Abela, C.: DAML enabled web service and agents in semantic web. In: Workshop on Web, Web Services and Database Systems, Lecture Notes Ccomputer Science (LNCS) (2003)
49. Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., Loser, A.: Super-peer-based routing strategies for RDF-based Peer-to-Peer networks. Journal of Web Semantics **1**(2), 177–186 (2004)

50. Ng, W.S., Ooi, B.C., Tan, K.L., Zhou, A.: PeerDB: A P2P-based System for Distributed Data Sharing. In: Proc. of the International Conference on Data Engineering (ICDE), pp. 633–644 (2003)
51. Prud'Hommeaux, E., Seaborne, A.: SPARQL query language for RDF. Working Draft WD-rdf-sparql-query-20061004, World Wide Web Consortium (W3C) (2006)
52. Pugh, W.: Skip lists: a probabilistic alternative to balanced trees. Communications of the ACM **33**(6), 668–676 (1990)
53. Rabin, M.: Fingerprinting by random polynomials. Technical report, CRCT TR-15-81, Harvard University (1981)
54. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: Proc. of ACM SIGCOMM, pp. 161–172 (2001)
55. Rhea, S., Kubiatowicz, J.: Probabilistic location and routing. In: Proc. of IEEE INFOCOM (2002)
56. Rompothong, P., Senivongse, T.: A query federation of UDDI registries. In: Proc. of International Symposium on Information and Communication Technologies (ISICT), pp. 578–583 (2003)
57. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale Peer-to-Peer systems. In: Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). Heidelberg, Germany (2001)
58. Sagan, H.: Space-filling curves. Springer-Verlag (1994)
59. Salutation Consortium: Salutation architecture specification version 2.0c. http://www.salutation.org (1999)
60. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: A scalable and ontology-based P2P infrastructure for semantic web services. In: Proc. of International Conference on Peer-to-Peer Computing (P2P) (2002)
61. Schmidt, C., Parashar, M.: Enabling flexible queries with guarantees in P2P systems. IEEE Internet Computing **8**(3), 19–26 (2004)
62. Schmidt, C., Parashar, M.: Peer-to-Peer approach to web service discovery. In: WWW: Internet and web information systems, vol. 7, pp. 211–229 (2004)
63. Sperberg-McQueen, C.M., Bray, T., Maler, E., Paoli, J., Yergeau, F.: Extensible markup language (XML) 1.0 (fourth edition). W3C recommendation, W3C (2006). Http://www.w3.org/TR/2006/REC-xml-20060816
64. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable Peer-to-Peer lookup protocol for Internet applications. IEEE/ACM Transaction on Networking (TON) **11**(1), 17–32 (2003)
65. Sun Microsystems: Jini Technology Core Platform Specification (2000). URL `http://www.sun.com/jini/specs/`
66. Tang, C., Dwarkadas, S.: Hybrid global-local indexing for efficient Peer-to-Peer information retrieval. In: Proc. of the Symposium on Networked Systems Design and Implementation (NSDI) (2004)
67. Tang, C., Xu, Z., Mahalingam, M.: pSearch: information retrieval in structured overlays. ACM SIGCOMM Computer Communication Review **33**(1), 89–94 (2003)
68. Tsoumakos, D., Roussopoulos, N.: Adaptive probabilistic search for Peer-to-Peer networks. In: Proc. of International Conference on Peer-to-Peer Computing (P2P) (2003)
69. UDDI Consortium: UDDI Technical White Paper (2002). URL `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`
70. Uschold, M., Gruninger, M.: Ontologies: Principles, methods and applications. Knowledge Sharing and Review **11**(2) (1996)
71. Yang, B., Garcia-Molina, H.: Improving search in Peer-to-Peer networks. In: Proc. of International Conference on Distributed Computing Systems (ICDCS) (2002)
72. Zhao, B., Huang, L., Stribling, J., Rhea, S., Joseph, A., Kubiatowicz, J.: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications (JSAC) **22**(1), 41–53 (2004)