Improving querying in Peer to Peer Systems

Maria-Del-Pilar Villamil, Claudia Roncancio, Cyril Labbé

Grenoble University / LSR-IMAG Laboratory
BP 72, 38402 St. Martin d'Hères, France
e-mail: Firstname.Lastname@imag.fr

Abstract. This paper presents the design, implementation and performance evaluation of a querying and indexing middleware for Distributed Hash Table (DHT) P2P systems. Data sharing in such systems is improved by supporting declarative location queries and other data management facilities. Queries may be conjunctions and disjunctions of conditions including comparison terms (e.g. $=, <, >$, *like* operators).
Unlike some current works, our proposal respects a true P2P architecture, does not rely neither on centralized catalogs nor on super peers, does not require hashing functions with particular properties and can be used with any current DHT P2P system. Our proposal on meta-data fragmentation, duplication and indexes leads to an efficient support of queries and allows a variety of execution plans. The scalability of the proposal is shown by experiments with our prototype. A qualitative and quantitative comparison with related works is presented.
Keywords: P2P systems, distributed query evaluation, large scale distributed systems.

1 Introduction

Peer-to-peer (P2P) is an emerging paradigm to provide large scale distributed systems. Their architecture does not suppose centralized servers and considers participating computers (peers) as having equivalent roles. Current P2P systems provide high flexibility in managing large sets of heterogeneous peers: systems are very dynamic, members may join and leave them easily. Two generations of P2P systems may be distinguished: the first one (e.g. Gnutella [1]) manages dynamic unstructured sets of participants. Object location requests submitted by users are propagated in the system by using flooding strategies [2, 3]. The second generation are structured P2P systems (e.g. CAN [4], Chord [5],Pastry [6]) proposing a logical organization of peers supported by a Distributed Hash Table (DHT) [7]. Such a DHT insures scalability and dynamicity of P2P systems and improves the propagation of object location requests: the number of contacted peers is reduced in comparison with the first generation. Important results are now achieved in providing flexible management of large and dynamic sets of peers. Nevertheless, from a data management point of view, functions provided on P2P systems remain restricted.

This work concerns querying in DHT based P2P systems. Contrary to first generation of P2P systems, these systems provide **comprehensive** answers [8] (i.e. answers containing all relevant data available in the system) to object location requests. Nevertheless, in their basic configuration, they do not provide

keyword search functions or multi attributes and comparative queries. They only offer access to objects giving their key. More powerful queries are difficult to handle since objects distribution criteria among peers is based on their key. Therefore the routing process, based on such keys, cannot be used to solve semantic queries. Several attempts to improve querying capabilities have been performed recently. Most of them concern location queries. Main approaches use meta-data to add semantics to stored data.

This paper introduces PinS, a middleware to improve data access and querying on top of DHT systems. It applies to a general context without hypothesis on the type of data stored in the system and considers attributes associated to objects as meta-data. This paper presents PinS design, implementation, performance evaluation and a comparison with existent propositions about object registration and query processing. [1]

Unlike some current proposals [10, 11], PinS respects a true P2P architecture, does not rely on centralized catalogs nor on super peers [12], does not require hashing functions with particular properties and can be used with any current DHT P2P system. PinS provides several execution strategies (others can be added) with different performance characteristics and enables query optimization based on the system characteristics and on the required QoS. Our prototype experiments show PinS good performances. PinS design choices are so that the number of objects registered in the system does not affect the complexity of the evaluation of conjunctive queries. This property is highly appreciable in large scale systems. PinS 1.0 is operational. Experiments have been performed on top of Pastry [13] in a system involving 1000 peers.

The paper is organized as follows. Section 2 reviews main features of DHT systems and points out functions used by our proposal. Section 3 introduces the PinS middleware. Section 4 presents the general execution strategy for queries composed of equality terms. Section 5 focuses on queries including comparison terms (e.g using $<, >$, *like* operators). Section 6 presents the architecture and performance measures of PinS 1.0. Section 7 analyzes related works whereas 8 concludes and presents future work.

2 Background on structured P2P systems

Structured P2P systems manage dynamic and heterogeneous sets of participant peers. These systems adopt a logical organization of peers supported by a (DHT) [7]. A hash table is used as index to improve data access. A hashing function associates a position to a key.[2] Key searches are therefore very efficient. A P2P system can be compared to a bucket array where a peer is a storage unit (bucket). This simile is used by structured P2P to enhance the management of large sets of peers. Structured P2P systems are therefore known as Distributed Hash Table Systems (DHTS). As a matter of fact, DHTS are not yet standardized but there are efforts on this direction [14, 15].

[1] Preliminary ideas on this work were introduced in [9]. No global proposal, no implementation and no experiments are presented there.

[2] In this paper, "key" denotes a value returned by the hash function.

In DHTS, locating an object is reduced to routing to the peers hosting the object. The main differences between DHT systems is routing geometry. CAN [4] uses a d-dimensional Cartesian space, Chord [5] and Pastry [6] a ring and Viceroy [16] a butterfly network. To find an object users have to know its key. Queries are called **location queries**. Its Answer contains the peer identifiers where the object is stored. The requester has to get then the object at one of these peers by himself. For more information about P2P systems see [12].

Our point of view is that the main functionalities of DHTS and systems using them tightly may be classified in two layers: **Distributed Lookup Service (DLS)** and **Distributed Storage Service (DSS)**. DLS layer provides efficient mechanisms to find peers (named by keys) by managing the routing information. Its main functions in our context is `lookup(key)`: returns the physical identifier of the peer in charge of a key.

DSS layer does the stored object administration (e.g. insertion, migration). Objects migrate from a peer that leaves the systems to one of its neighbors to insure its durability. Such systems offer load balancing and caching strategies. The main external functions of DSS layer are: `get(key)`: returns the object(s) identified by `key`. `put(key, Object)`: inserts in the system an object identified by its key. Both functions use `lookup` to locate the peers responsible for the key. Examples of systems in DSS layer are Past [17] and DHash [18].

3 Introduction to the PinS Middleware

PinS offers data/meta-data registration and location queries on top of DHT P2P systems. Data may be of any type[3]. PinS works on meta-data and data types are not relevant. Meta-data is composed by <attribute,value> tuples (e.g. *Year=1988*). Registration and querying facilities are presented here.

3.1 Data and meta-data registration and management

Registering an object also includes the creation of **local catalog** and **location data** meta-data. Local catalog keeps information detailed about objects stored in a peer and Location data is based on the given <attribute,value> information[4]. PinS uses the hash function provided by the DSS, `DSSHashF()` in the following, to decide location, object and meta-data identifiers.

Let's consider the registration of the book entitled *Tala* written by *Gabriela Mistral* (GM in the following), illustrated in figure 1. The process begins in peer P600 in charge of the data/meta-data registration process. This peer obtains the object identifier IdObj1(123 in figure 1) calling `DSSHashF(O1)`. This information is used by DSS to decide the peer in charge of the object storage. A local catalog record is created in the same peer that store the object, see point 1.b in figure 3. This record contains all meta-data of the object: term identifier, attribute name, attribute value and a list of local objects satisfying the term. Peers who store objects and local catalogs have the role of **storage peer**. The same process is followed to create the location data. Each term Ti composing O1's meta-data is handle as an object: PinS obtains term identifier IdTi using `DSSHashF(Ti)` (e.g.

[3] images, video, text, software components,etc

[4] Attribute names are supposed to be accepted by the community using the system.

510 is the identifier of term *Date=1938*), and it is used to identify the peer in charge of location-data storage. A location data record is composed by the term identifier and all object identifiers register in the system that satisfy this term. DSS registers location data calling `put(IdTi, IdObj1)`, see points 2 in figure 3. Peers responsible for the location data storage play the role of **location peer**.

At the end of the registration process all object information is fragmented and distributed in the P2P system in location data records and replicated partially using the local catalogs records. This way to handle data offers a scalable solution and enable the use of different strategies to evaluate queries.

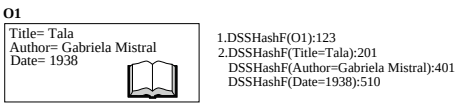

Fig. 1. Example of object description

Query ::= Term [and | or Term]*
Term ::= EqTerm | IneqTerm | LikeTerm
EqTerm ::= AttributeName = value
IneqTerm ::= AttributeName op value
op is <, >, =< or >=
LikeTerm ::= AttributeName like ⌈%⌉char+⌈%⌉

Fig. 2. Query Language

3.2 Querying facilities

With PinS a **location query** is a declarative query with conjunction and disjunction of terms of the form "attribute operator value" (e.g. *Year=1988 and Title like "On%"*). Figure 2 shows the grammar of the language. Attributes are those declared at object registration time. The operator used in a term can be $=$, *like* or $\leq, \geq, <, >$. Terms using these operators are called **EqTerm**, **LikeTerm** and **IneqTerm** respectively. PinS implements several query execution strategies using standard functions provided by the underlying DSS and DLS layers. Processing queries involving IneqTerms or LikeTerms (e.g. *Year* $>$ *1987*) is time consuming in large scale P2P systems. PinS proposes a solution efficient for such queries. Any peer may receive location queries. The contacted peer, referred as **access peer** in the following, initiates the query evaluation process.

4 Supporting queries with EqTerms

Location queries evaluation depends on the nature of terms they involve. This section concerns the support of queries composed exclusively by EqTerms.

The *General EqTerm (GE) Strategy* relies on the use of the hash code of the terms composing the requested query (see figure 4) as typical in P2P systems. All terms Ti composing the query are translated to their term identifier IdTi using the `DSSHashF(Ti)` function. For example, in figure 1, 510 is the IdTi of *Date=1938*.

IdTi's are used as key to obtain all location data related to the term i.e. calling `get(IdTi)` – see figure 4, line 3. Using the returned location data, the access peer finishes the process calculating query's terms conjunctions with intersections and terms disjunctions with unions – See lines 8-11 in figure 4. The set of returned object identifiers is a comprehensive answer.

Local catalogs (LC) offer an alternative to evaluate Eq/Ineq/Like terms. The use of these catalogs combined to the GE strategy allow some kind of load balancing to avoid overloading location peers of frequently asked terms. Some

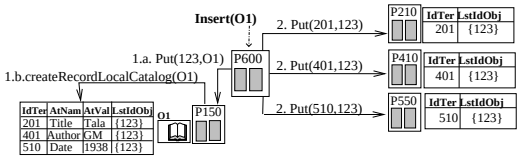

Fig. 3. Registration process

```
executeGeneralStrategy(lstEqTerms)
(1)  for each EqTerm in lstEqTerms loop
(2)      idTerm = DSS.calculateHashId(EqTerm.Value)
(3)      ans[i] = DSS.get(idTerm)
(4)      numEqTerms++
(5)  end loop
(6)  ansQuery = initAnswer()
(7)  for i in 1 .. numEqTerms loop
(8)      if (lstEqTerms[i].Operator == CONJUNCTION)
(9)          intersection(ansQuery, ans[i])
(10)     else if (lstEqTerms[i].Operator == DISJUNCTION)
(11)         union(ansQuery, ans[i])
(12) end loop
(13) return (ansQuery)
```

Fig. 4. GE strategy algorithm

terms of the query are evaluated with the GE strategy, the others using local catalogs. The execution happens as follows. A set of object identifiers is obtained as answer to the part of the query evaluated with the GE strategy. The system then contacts the storage peers of such objects to evaluate the remainder part of the query by using their local catalogs. The answer is then retrieved to the **coordinator peer**, peer responsible for the decomposition of the initial query and who composes the final answer.

Several execution plans can be elaborated. Local catalogs can be used or not. In general, the most selective terms are preferred to be evaluated with the GE strategy. Furthermore, the access peer plays naturally the role of coordinator but this responsibility can be delegated to another peer involved in the query evaluation (a location peer or a storage peer). The distribution of this responsibility is also possible and may be interesting when intermediate results are big.

To finish, notice that replication and load balancing strategies provided by the underlying DSS operate independently but contribute to the global solution.

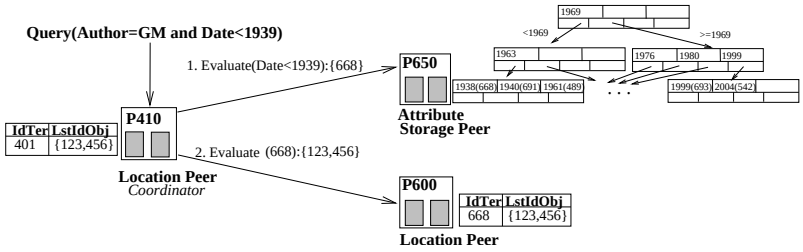

Fig. 5. Query processing with IX strategy.

5 Supporting queries with IneqTerms

Queries involving comparison terms (e.g. $<, like$) are helpful in numerous situations but their support is not straightforward in DHTS. This section proposes two execution strategies to provide Ineq/Like Term queries: Local catalog (**LC**) and Index (**IX**) based strategies.

5.1 Local Catalog strategy

LC strategy is a variant of the one introduced in section 4. Let's consider the query *Author=GM and date<1939*. The location peer of term *Author=GM* –

peer P410 in figure 5 – is chosen as query coordinator. It uses the *Author=GM* term identifier to obtain locally the identifiers of objects satisfying the term. A list of object identifiers is returned (e.g. 123 and 456 in figure 5). The query coordinator uses these object identifiers to identify the storage peers to contact to continue query execution e.g. peers P150. The LC strategy applies to queries with at least one EqTerm. However, for queries with no EqTerm this strategy contacts all storage peers of the system. As this is unacceptable, PinS proposes the Index (IX) based strategy.

5.2 Index based strategy

IX strategy improves the support of Ineq/Like terms by using index structures on attributes with ordered domains (e.g. Date). Indexes store attribute values and term identifiers allowing to identify the location peers to contact to evaluate an Ineq/like term. This allows an execution strategy where only peers with objects satisfying the Ineq/Like term are contacted. Indexes are logical, do not contain peer identifiers, nor peer addresses, are independent of objects location and object's migration do not invalidate them. Let's consider the Date attribute, see figure 5. Its index is created on the peer related to attribute Date according to DSSHashF (peer returned by `lookup(DSSHashF(Date))`). It is called **date's attribute storage peer** from now on.

IX strategy uses B+trees [19] but other data structures can be used. They are handled with standard search, insert and delete B+tree algorithms. A B+tree leaf node is composed by attribute's values (e.g. 1938) and identifiers of terms using these values (e.g. `DSSHashF(Date=1938)=668`), see figure 5. Terms identifiers "represent" objects in the system satisfying the term (e.g. *Date=1938*). To evaluate the query $Date < 1939$, PinS contacts the *Date*'s attribute storage peer by calling `lookup(DSSHashF(`*Date*`))` – P650 in figure 5. This peer searches locally all values satisfying the query, to identify the location peers responsible of the corresponding IneqTerm storage (in figure 5, value 1938). Finally, the coordinator peer contacts the location peer to continue the evaluation process described in section 4. The IX strategy applies to queries with at least one Ineq/Like Term. The DSS replication mechanism can be used to reduce a possible overload on attribute storage peers.

Notice that an index have to be updated only if a not already existing attribute value is introduced or a value disappear. When this happens, the location peer in charge of the related term (e.g. Date=1969) sends an update event to the attribute storage peer who updates the index.

6 Implementation and measures

PinS 1.0 has been developed in Java 1.4.2 and is operational on Linux. The PinS layer provides data/meta-data registration and query processing as proposed in the preceding sections. Each peer in the system has the three layers: DSS, DLS and PinS to provide all functionality.

6.1 PinS 1.0 software architecture

The PinS layer considers DSS and DLS layers as separate components (see section 2). PinS mainly uses `DSS.calculateHashId(Obj)`, `DSS.get(key)`, `DSS.-`

`put(key, Object)` and `DLS.lookup(key)` functions. The `DSS` and `DLS` components can be implemented using different DHT systems. The current implementation relies on FreePastry 1.3.2 [13] who provides Past at the level of DSS and Pastry at the level of DLS. In the following the components of the PinS layer are briefly introduced.

Object registration: Object registration includes the insertion of data/meta-data as described in section 3.1. The two fundamental issues, data placement and data storage, are insured by `ObjectMgr`, `MetadataMgr` and `CommunicationMgr` components. `ObjectMgr` provides object storage functions, it uses `DLS.lookup` and `DSS.put` functions to make data placement and data storage and interacts with the `MetadataMgr`. `MetaDataMgr` creates and handles information related to meta-data (e.g. location data) and indexes (considered as meta-data). It also uses `DSS` and `DLS` components. `CommunicationMgr` provides communication between the PinS layer of different peers in the system. It optimizes communication costs by reducing the number and size of messages exchanged in the system.

Location Query support: Query evaluation is implemented by the `QueryMgr` . It insures query decomposition and optimization. `QueryMgr` differentiates queries composed exclusively by EqTerms from those involving IneqTerms. For each case, it implements the execution strategies presented in the preceding sections.

6.2 PinS performance measures

Experiment results demonstrate the feasibility of our proposal and good performance characteristics. One of the objectives of this implementation was to confirm the theoretical analysis presented in [9] (see figure 6). Executions were performed on 38 nodes of the *i-cluster2* cluster (of the IMAG-ID laboratory, http://i-cluster2.inrialpes.fr), with Itanium-2 processors 64 bits at 900 MHz, 3Gb memory. We deployed 1000 PinS peers and considered books stored in the DHT P2P system. Meta data is composed by five attributes – *author, title, abstract, YearEdition, YearPublication*. Figure 7 shows the evaluation of queries: **Q1**:$Author$ = *"Molière" and YearEdition* > *1640* and **Q2**:$Author$ = *"Corneille" and YearEdition* < *1640* –execution time according to the number of object stored in the system. Both queries were evaluated using LC and IX strategies. The selectivity factor of Q1 is 0.02. The effective cardinality of the IneqTerm (|*YearEdition* > *1640*|) is 40. The effective cardinality of $Tineq_i$ is the number of distinct values of objects registered in the system satisfying the condition. Q2 has 0.2 as selectivity factor and 10 as the effective cardinality of the IneqTerm. Experimental results (figure 7) confirm our theoretical analysis and that execution time for IX strategy is not affected by the number of objects stored in the system.

7 Related work on querying in DHT P2P systems

Related works are briefly discussed according to the expressive power of the queries. A quantitative analysis is also presented.

Qualitative analysis: KSS[18] supports conjunctions of EqTerms on top of the DHash/Chord P2P systems. They improve query processing by replicating the whole meta-data of objects on all peers supposed to store a part of such meta-data. Unlike PinS, KSS does not offer terms disjunctions nor IneqTerms

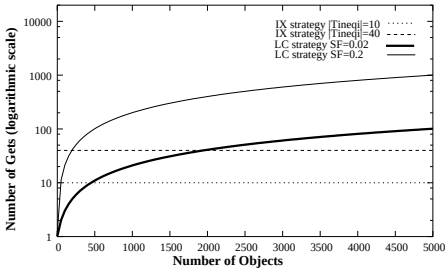

Fig. 6. Number of Gets versus number of objects

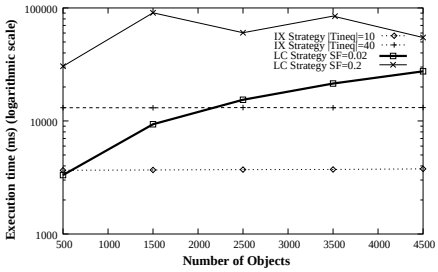

Fig. 7. Execution time versus number of objects using 1000 peers

and does not works with any typical underlying DSS. MLP[20] also supports EqTerm queries and introduces peer's group hierarchies based on peers location. From our point of view this aspects are at the DSS level. The existence of such groups facilitates parallel processing of queries. [18] and [20] adopt both some pre-calculated queries solution. However, this solution is space consuming. Table 1 compares different works that provide Ineq/Like Term evaluation. MAAN[11] uses an order-preserving hash function. This requires the initial attribute distribution (e.g. Exponential, Normal) to distribute uniformly data among peers. PinS does not require a function with particular characteristics. RangeGuard[21]

	Hypothesis	Additional routing information	Query langage	Optimization	Query evaluation
MAAN [11]	Order preserving hashing function. attribute distribution is known	For each ordered hashing function	IneqTerm	Replication of local catalogs on location peers	Based on successor function.
RG[21]	RG layer as Super peers [12]. Known minimal and maximal values of attributes	On each peer in range guard layer	IneqTerms Join, order by, group by	Replication of local catalogs in range guard layer	Based on range guard layer.
[22]	XML documents. Peers physical address in location data	No	XPath	Fragmentation and duplication of meta-data using an XML hierarchy	-
PinS	No	No	IneqTerms, LikeTerms	Local catalogs. Indexes on attributes	Only relevant peers are contacted.

Table 1. Works that provide Ineq/Like Terms evaluation.

approach requires the knowledge of the minimal and maximal values stored in the system. A uniform distribution of data on RG peers is not guaranteed. The MAAN and RG evaluation process is based on the use of the **successor**[5] function. This function is difficult to be defined in DLS using routing geometries as butterfly (Viceroy) or d-dimensional (CAN). In addition, peers with no relevant data are contacted. In [22] location data includes peer physical address. This choice increases the cost of the update process when a peer leaves the system and can be a problem if the system is very dynamic like typical P2P systems.

Quantitative analysis: Table 2 compares PinS with other works that provide IneqTerms's evaluation (i.e. MAAN and RG). The element of comparison is the number of peers contacted to evaluate a query composed of l IneqTerms

[5] function to contact the next neighbor of a peer using a defined order e.g. peers identifiers.

$Q = Tineq_1$ *and ... and* $Tineq_l$, where $Tineq_j = i \leq A_i \leq h$. Table 2 uses

	Number of peers contacted to evaluate Q
MAAN Iterative	$l * (LogN + Min(\|Tineq'_i\|, N))$
MAAN SAD	$LogN + Min(\|Tineq'_i\|, N)$
RG	$l * (Log(N - N_{RGP}) + Min(\|Tineq'_i\|, N_{RGP}))$
PinS IX	$LogN * (l + \sum_{i=1}^{l} \|Tineqi\|)$

Table 2. Number of peers contacted to evaluate Q using MAAN, RG and PinS IX strategies

Number of peers (N)	**Percentage limit (PL)**
10^2	50
10^3	33
10^4	25
10^5	20
10^6	16

Table 3. Comparison between PinS and MAAN

the following variables: N_{RGP} the number of peers in the range guard layer to RG approach and $|Tineq'_i|$ the theoretical cardinality of $Tineq_i$ – the number of distinct values for A_i satisfying the condition. Table 2 analysis allows to define the percentage of value used for one attribute: $PL = |Tineq|/|Tineq'|$. Table 3 shows PL value where the number of contacted peers is the same for PinS IX and MAAN iterative. PinS is better when $|Tineq|/|Tineq'| < PL$, otherwise MAAN iterative is better. For a large scale system PinS is more efficient when an attribute has a large range of values and few of them are used.

8 Conclusions and research perspectives

DHT based P2P systems provide appropriate management of large scale dynamic peers. They are ideal for key searches but don't provide, in their basic configuration, a high level query language to find stored data. Recent proposals improve this feature, but most of them require special hashing functions (e.g. order preserving functions) or rely on distinguished peers. PinS proposes a set of functionalities improving data sharing on DHT P2P systems without such constraints. Meta-data, sets of attributes and their values, are associated to shared objects. Location queries may be conjunctions and disjunctions of terms (attribute operator value) using the $=, \leq, \geq$ and *like* operators. PinS adapts database techniques to enable the use of different query evaluation strategies. It uses indexes in an original way to support efficiently comparative queries.

PinS is independent of the underlying P2P routing structure. To clarify levels of interaction with the underlying DHT system, we distinguished the Distributed Storage Service and the Distributed Lookup Service. PinS has been implemented in Java and experimented using FreePastry 1.3.2 for the underlying DHT based P2P system. Experiments with up to 1000 peers have been performed. They confirm PinS scalability and show that the execution time is not affected by the number of objects stored in the system when using our IX strategy.

PinS design choices (e.g. meta-data management) allow the addition of execution strategies. These elements could be used to perform a context aware query optimization considering current system configuration (e.g. number of peers, storage) and behavior (e.g. data distribution, query types). Ongoing work concerns this point and more experiments in large scale configurations. Index fragmentation and caching strategies related issues should also be further investigated.

References

1. Gnutella: The Gnutella Protocol Specification v0.41. (http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf)
2. Yang, B., H.Molina: Efficient search in peer-to-peer networks. Technical report, http://dbpubs.stanford.edu:8090/pub/2001-47 (2001)
3. Q.Lv, P.Cao, E.Cohen, K.Li, S.Shenker: Search and replication in unstructured peer-to-peer networks. In: Proc. Int'l Conf. on Supercomputing. (2002)
4. S.Ratnasamy, P.Francis, M.Handley, R.Karp, S.Shenker: A Scalable Content Addressable Network. In: ACM SIGCOMM. (2001)
5. I.Stoica, R.Morris, D.Karger, F.Kaashoek, H.Balakrishnan: Chord: A Scalable "Peer-To-Peer" Lookup Service for Internet Applications. In: ACM SIGCOMM. (2001)
6. A.Rowstron, P.Druschel: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. LNCS-2218 (2001)
7. M.Harren, J.Hellerstein, R.Huebsch, B.Loo, S.Shenker, Stoica, I.: Complex queries in dht-based peer-to-peer networks. IPTPS (2002)
8. N.Daswani, H.Garcia-Molina, B.Yang: Open Problems in Data-Sharing Peer-to-Peer Systems. In: Proc Int'l Conf. on Database Theory. (2003)
9. M.Villamil, C.Roncancio, C.Labbe: PinS: Peer to Peer Interrogation and Indexing System. In: Proc. IDEAS. (06/2004)
10. I.Brunkhorst, H.Dhraief, A.Kemper, W.Nejdl, Wiesner, C.: Distributed Queries and Query Optimization in Schema-Based P2P Systems. IPTPS (2003)
11. M.Cai, M.Frank, J.Chen, P.Szekely: MAAN: A Multi-Attribute Addressable Network for Grid Information Services. Proc. of Int'l WS on Grid Computing (2003)
12. B.Yang, H.Molina: Designing a Super-peer Network. Proc. of IEEE Int'l Conf. Data Engineering (2003)
13. FreePastry: Rice University. (http://freepastry.rice.edu/FreePastry/)
14. F.Dabek, B.Zhao, P.Druschel, I.Stoica: Towards a common API for structured peer-to-peer overlays . IPTPS (2003)
15. F.Dabek, E.Brunskill, M.Kaashoek, D.Karger, R.Morris, I.Stoica, H.Balakrishnan: Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. In: HotOS-VIII. (2001)
16. D.Malkhi, M.Naor, D.Ratajczak: Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In: ACM Symposium on Principles of Distributed Computing. (2002)
17. A.Rowstron, P.Druschel: Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In: Symposium on Operating Systems Principles. (2001)
18. O.Gnawali: A Keyword-Set Search System for Peer-to-Peer Networks. Master thesis, Massachusetts Institute Of Technology (2002)
19. H.Molina, J.Ullman, J.Widom: Database system implementation. Prentice Hall (2000)
20. S.Shing, G.Yang, D.Wang, J.Yu, S.Qu, M.Chen: Making Peer-to-Peer Keyword Searching Feasible Using Multi-level Partitioning. In: Proc. of IPTPS. (2004)
21. P.Triantafillou, T.Pitoura: Toward a unifying framework for complex query processing over Structured Peer-to-Peer Data Networks. VLDB WS on Databases, Information Systems, and Peer-to-Peer Computing (2003)
22. L.Galanis, Y.Wang, S.Jeffery, D.DeWitt: Locating Data Sources in Large Distributed Systems. In: Proc. of Int'l Conference on VLDB. (2003)