ISSN:2231-0711
www.ijcset.net

Searching Multidimensional Historical Data Using QuiSea Algorithm in P2P Networks

D.Raghava Lavanya[1], Y.A.Siva Prasad[2]

Department of computer science and technology, KL University
Green Fields, vaddeswaram, Guntur, Andhra Pradesh, India
1draghavalavanya@gmail.com
2 sivaprasady@gmail.com

Abstract: **Compressing and creating index to large database may result into time consuming. In Peer-to-peer framework, network may have millions of nodes, so creating index to large database requires number of index keys. To overcome this disadvantage and to simplify searching process we introduce QueSea algorithm. This algorithm first selects some part of network and then starts searching. In existing system, we have flooding and random Walk approaches to search multidimensional data, these approaches can be used in Peer-to-peer networks.**

Key Terms: Searching Algorithm, performance analysis, multidimensional data management, indexing, compression.

1 INTRODUCTION:

In unstructured peer-to-peer (P2P) networks, each node does not have global information about the whole topology and the location of queried resources. Because of the dynamic property of unstructured P2P networks, correctly capturing global behavior is also difficult [1], [2]. Search algorithms provide the capabilities to locate the queried resources and to route the message to the target node. Thus, the efficiency of search algorithms is critical to the performance of unstructured P2P networks [3]

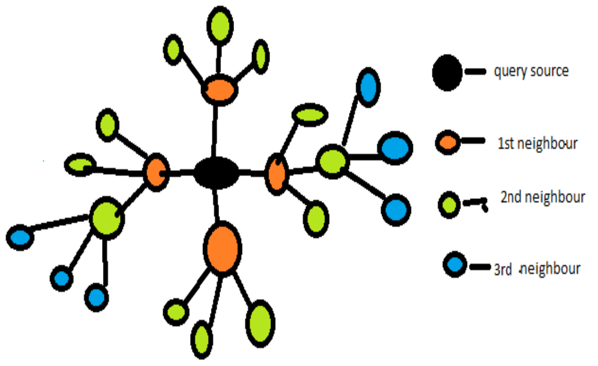


Figure 1:**simple scenario of p2p network to demonstrate the operation of flooding and random walk**

Previous works about search algorithms in unstructured P2P networks can be classified into two categories: Breadth first search (BFS)-based methods, and Depth first search (DFS)-based methods. These two types of search algorithms tend to be inefficient, either generating too much load on the system [4], [5], or not meeting user requirements [6].

Flooding, which belongs to BFS-based methods, is the default search algorithm for Gnutella network [7], [8]. By this method, the query source sends its query messages to all of its neighbors. When a node receives a query message, it first checks if it has the queried resource. If yes, it sends a response back to the query source to indicate a query hit. Otherwise, it sends the query messages to all of its neighbors, except for the one the query message comes from. The drawback of flooding is the search cost. It produces considerable query messages even when the resource distribution is scarce. The search is especially inefficient when the target is far from the query source because the number of query messages would grow exponentially with the hop counts. Fig. 1 illustrates the operation of flooding. The link degree of each vertex in this graph is 4. If the network grows unlimited from the query source, the number of query messages generated by flooding at each hop would be 4, 12, 36, . . . , respectively. If the queried resource locates at one of the third neighbors, it takes 4 +12+ 36 = 52 query messages to get just one query hit.

2 RELATED WORKS

2.1 About Searching

Flooding and Random Walk are two typical examples of blind search algorithms by which query messages are sent to neighbors without any knowledge about the possible locations ~~of~~ the queried resources or any preference for the directions to send. Some other blind search algorithms include modified BFS (MBFS) [23], directed BFS [6], expanding ring [17], and random periodical flooding (RPF) [24]. These algorithms try to modify the operation of flooding to improve the efficiency. However, they still generate a large amount of query messages. Jiang et al. propose a LightFlood algorithm, which is a combination of the initial pure flooding and subsequent tree-based flooding [25], [26].Two topics are strictly related to our work: the compression of multidimensional data and the management of multidimensional data in P2P networks.

2.2Compression of Multidimensional Data

The problem of effectively summarizing multidimensional data into lossy synopses has been investigated mainly in the contexts of query optimization

[38] and exploratory OLAP analysis [6]. Some techniques are said to be parametric, as they rely on the assumption that data are distributed according to a mathematical (either statistical or polynomial) model. In particular, wavelet-based techniques (which are the most investigated in this group) work in two steps. First, a wavelet transform is applied to the data, yielding a set of coefficients. Then, these coefficients are suitably filtered and the "most relevant" ones are kept. Applying the inverse wavelet transform to these coefficients results in an approximate reconstruction of the data.

3 OPERATION OF QUISEA ALGORITHM

There are main 4 steps in algorithm
Step-1: Data is to be compressed and indexed.
Step-2 : Data should be distributed.
Step-3: Searching process will be started based on number of hops.
Step-4: After method is selected required multidimensional data will be returned from node.

3.1 Partitioning the Data Domain

The aim of the partitioning step is to divide the data domain into nonoverlapping blocks. These blocks will be compressed separately, yielding distinct subsynopses. For each of them, a portion of the amount of storage space B chosen to represent the whole synopsis will be invested. The distribution of B among blocks will take into account the following requirements:
B must be fairly distributed among blocks: The assignment of different amounts of storage space to the blocks for representing their subsynopses should depend on the differences in homogeneity among the blocks. Intuitively enough, the more homogeneous the data inside a block, the smaller the amount of information needed to effectively accomplish its summarization. Each block must be assigned a "small" portion of B: The subsynopses over the blocks are the data that will be hosted by peers and exchanged across the P2P network. As explained above, building subsynopses with "large" size would impose a significant constraint on the amount of storage space which should be made available by each peer. On the contrary, defining small-size subsynopses results in limiting the storage and computational resources required at each peer for storing and querying data, as well as reducing both the download and upload traffic needed for supporting data exchange.
We denote the maximum amount of storage space which can be invested for summarizing a single block as Bmax. In our prototype, we set Bmax ¼ 256 KB: this is the threshold value which proved effectiveness in several file sharing applications (such as Gnutella itself) for limiting the download segment size, i.e., the size of the atomic file portions exchanged among peers.

In order to satisfy the afore mentioned requirements, the partitioning of the data domain and the distribution of the storage space are accomplished according to the following iterative scheme. We start from a partition consisting of a single block (corresponding to the whole data domain) which is assigned the overall amount of storage space B. At each step, a block b_ of the partition is chosen and its range is split into two subranges: two new blocks b0 and b00 are created (corresponding to the MBRs1 of these subranges) and the partition is refined by replacing b_ with b0 and b00. Then, the distribution of B among blocks is updated accordingly. The partitioning ends when every block is assigned an amount of storage space which does not exceed Bmax. In Fig. 2, a partitioning of a 2D data population is shown (dashed line boxes represent the MBRs of the blocks). We now explain the criteria adopted to select and split blocks, and the strategy for distributing B among the blocks of a partition.

3.1.1 Selecting and Splitting Blocks of the Partition

At each step, the least homogeneous block b_ in the current partition is split. The homogeneity of a block b of a data population D is measured by evaluating the Sum Squared Error of its data population, that is, SSEðbÞ ¼ Pi2bðb½i_ _ bÞ2, where i 2 b means that i is a point (i.e., a set of multidimensional coordinates) inside the range of b, D½i_ is the value associated with point I of D, and b is the average value in b. After b_ is selected, the split is performed which results in the pair of blocks b0, b00 having the most similar SSE with respect to all the possible splits of b_ performed along every dimension.
Observe that splitting a block always results in a pair of blocks more homogeneous than the original one (in fact, it is well known that SSE is superaddictive, as SSEðb_Þ _ SSEðb0Þ þ SSEðb00Þ). Hence, employing this splitting strategy aims at creating blocks with similar degrees of homogeneity, while refining the partition toward more and more homogeneous blocks. Specifically, having blocks with similar homogeneity is likely to yield subsynopses with similar accuracy, while having blocks as much homogeneous as possible is likely to enhance the accuracy of each subsynopsis.

3.1.2 Distributing B among the Blocks of the Current Partition

First, a fixed portion Bmin of B is assigned to every block (the meaning of Bmin will be clearer in the following), and then, the remainder of B is distributed on the basis of the homogeneity of the blocks. That is, if the current partition consists of k blocks b1; . . . ; bk, then each bi is assigned the following amount of storage space:
BðbiÞ ¼ Bmin þ
SSEðbiÞ
Pk
j¼1 SSEðbjÞ
_ ðB _ k _ BminÞ:
The value of Bmin is the amount of space needed to store the most compact representation of a block according to the compression technique adopted. For instance, if a

histogram is employed, Bmin is the space consumption of representing both the range of the block and a set of aggregate values (such as the sum) summarizing the data inside a block. Basically, assigning at least Bmin to each block means that every block of the partition is guaranteed to be represented in the overall data synopsis.

1. The minimum bounding rectangle (MBR) of a range r is the minimum subrange of r containing all non-null data in r.

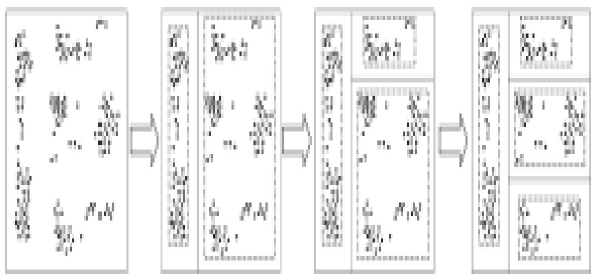


Fig. 2. Partitioning a 2D data population.

The iterative algorithm accomplishing the partitioning is shown in Fig. 3. It uses a priority queue which, at each step, contains the blocks of the current partition, ordered by their SSE. Variable SSEtot stores the sum of the SSEs of the blocks in the current partition, whereas variables SSEmax and spacemax represent the SSE of the least homogeneous block of the partition (i.e., the block at the head of the queue) and the storage space assigned to it, respectively. Iteratively, the least homogeneous block is extracted from the queue and split into two new blocks, which are, in turn, inserted into the queue. After splitting a block, SSEtot is updated to take into account the overall SSE reduction due to the split, SSEmax is assigned the SSE of the new head of the queue, and spacemax is recomputed on the basis of the new values of SSEtot and SSEmax. The partitioning ends when spacemax reaches a value less than Bmax (obviously, since spacemax refers to the least homogeneous block of the partition, the fact that spacemax < Bmax implies that every block of the partition is assigned less than Bmax). The algorithm returns the partition of the input data population represented as a set of pairs hr; si, where r is the range of a block and s is the amount of storage space to be invested for its summarization.

3.2 Compressing Data Blocks

At this step, a suitable compression algorithm is run on each of the k pairs hb1; s1i; . . . ; hbk; ski resulting from the partitioning step, and subsynopses h1; . . . ; hk are obtained, where each hi is a compressed representation of bi consuming storage space si. Several compression techniques (such as histograms or sampling [23]) can be employed to accomplish data summarization, as both the partitioning strategy and the techniques used for distributing and querying the subsynopses (which will be described in the following sections) are orthogonal to this choice. Our prototype embeds CHIST [10], which has been shown to be very effective in constructing multidimensional histograms providing accurate estimates of range queries. Briefly, Clustering-based Histogram (CHIST) exploits a density-based clustering algorithm to construct a set of (possibly overlapping) blocks covering the nonempty portions of the data domain. For each block (called bucket, according to standard histogram terminology), its boundaries as well as some aggregate value summarizing its data are stored. In our current implementation, each bucket is associated with the result of evaluating the sum aggregate operator: this way, the summary data suffice to estimate range sum queries.

3.3 Indexing Compressed Data

At this step, an index is built on top of the subsynopses resulting from the compression step. This index will be exploited for locating the data involved in the queries across the network.

Since each subsynopsis can be viewed as a hyper rectangular object, any indexing technique suitable for spatial data can be employed at this step. Specifically, in our prototype, an aggregate R-Tree [22], [26] was used, which is an R-Tree augmented with some aggregate information supporting aggregate range queries. The objects inserted in the aggregate R-Tree are pairs hIDðhiÞ;MBRðhiÞi, where IDðhiÞ is the identifier assigned to the subsynopsis hi and MBRðhiÞ is the hyperrectangular range covered by hi. The index is first populated and then partitioned, in order to make it prone to be distributed across the network. These two steps are described in the following.

3.3.1 Creating the Index

In order to limit the traffic needed for the maintenance of the index after its distribution, a compact index is desirable. Since data are historical, the storage space consumption of the R-Tree can be reduced by adopting packing strategies for its construction, which aim at obtaining 100 percent space utilization in each node. In particular, we adopt the R-Tree-packing technique proposed in [27]. This was shown to perform better than other packing strategies in supporting region queries on rectangular data, which is the feature of interest in our case. Briefly, this technique works in three steps. First, the data domain is linearized according to a Hilbert space filling curve. Then, the MBRs of the subsynopses are sorted on the Hilbert values of their centers. Finally, the list of MBRs is packed in groups of cardinality f (where f is the R-Tree fan-out), and for each group, a node is created containing the MBRs in the group as well as the identifiers of the corresponding subsynopses. These nodes are the leafs of the R-Tree. Inner nodes are constructed by proceeding bottom-up on the tree level: the ith level is built by first sorting the nodes at the ði þ 1Þth level on ascending creation time, and then, packing this sequence of nodes in the same way as explained for the leaf level. This way, every MBR m in a node at the ith level covers the MBRs of a node n at the i þ 1th level, and n is referenced by m. Each MBR m in (both inner and leaf) nodes of the aggregate R-Tree is stored along with the sum sumðmÞ of the values lying inside it. As it will be clearer in the following, this enhances the query evaluation, as it reduces the number of peers to be accessed for computing query answers.

This strategy yields almost 100 percent space utilization and due to the good clustering properties of the Hilbert curves, it tends to create leafs referencing subsynopses whose MBRs are "close" to one another. The aggregate R-tree indexing the subsynopses will be denoted as I.

3.3.2 Partitioning the Index

After being populated, I is partitioned in "small"-size portions which are prone to be distributed across the network. The reason for partitioning the index is the same as for limiting the amount of storage space invested for a single synopsis, that is, distributing small-size index portions across the network prevents peers from being overloaded in terms of upload and download traffic needed for supporting index replication.

Our approach for partitioning I is similar to that proposed in [30], where a distributed version of the R-Tree (Master R-Tree) was introduced, and works as follows: First, I is divided into two portions Isup and Iinf , which consist of the inner nodes and the leafs of I, respectively. Then, Iinf is, in turn, subpartitioned into m leaf portions inf1, . . . , infm: each leaf portion infi contains a number of leafs in Iinf whose overall storage space consumption is Bmax. Specifically,
inf1, . . . , infm are obtained by sorting the leafs in Iinf on their creation time and grouping them into sequences of cardinality bBmax P c, where P is the size of the pages of I. After the creation of leaf portions, the pointers in Isup to single leafs of Iinf are replaced with identifiers of entire leaf portions.

From now on, we will refer to portions of the index and subsynopses generically as s-blocks.

Observe that due to the packing-based index creation, the order of leafs is determined by the Hilbert ordering of the MBRs inside them. Hence, each infi is likely to contain leafs whose MBRs are close to one another. This aims at reducing the number of leaf portions to be accessed in order to evaluate range queries, as the data involved in a range query belong to subsynopses whose MBRs are close to one another. As it will be clearer in the following, since the index will be distributed across the network by assigning inf1, . . . , infm to different peers, this strategy will result in reducing the number of peers to be accessed for locating the subsynopses involved in range queries. Fig. 3 shows the aforementioned index partitioning scheme. Unlike Iinf , the Isup s-block is not subpartitioned: in fact, its size is very small (in all practical cases, smaller than 256 KB).

The main difference of our approach from the Master RTree is the grouping strategy adopted at the leaf level. Briefly, in the Master R-Tree, Iinf is partitioned into portions containing chunks of data whose MBRs are far from one another, and this aims at parallelizing query evaluation as much as possible. We do not use this strategy as in our approach, index leafs do not contain data, but references to data: thus, employing the same grouping strategy as Master R-Tree would result in increasing the number of leaf portions to be accessed for locating the subsynopses involved in the queries, thus increasing network traffic. Indeed, in our framework, parallelization of query evaluation is obtained by distributing the subsynopses among different.

QuiSea is designed as a generalization of flooding, MBFS, and RW. There are two phases in QuiSea. Each phase has a different searching strategy. The choice of search strategy at each phase depends on the relationship between the hop count h of query messages and the decision threshold n of QuiSea.

3.3.1 Phase 1. When h _ n

At this phase, QuiSea acts as flooding or MBFS. The number of neighbors that a query source sends the query messages to depends on the predefined transmission probability p. If the link degree of this query source is d, it would only send the query messages to d _ p neighbors. When p is equal to 1, QuiSea resembles flooding. Otherwise, it operates as MBFS with the transmission probability p.

3.3.2 Phase 2. When h > n

At this phase, the search strategy switches to RW. Each node that receives the query message would send the query message to one of its neighbors if it does not have the queried resource. Assume that the number of nodes visited by QuiSea at hop h ¼ n is the coverage cn, and then the operation of QuiSea at that time can be regarded as RW with cn walkers. However, there are some differences between QuiSea and RW when we consider the whole operation. Consider the simple scenario shown in Fig. 1. Assume that the decision threshold n is set as 2. When h > 2, QuiSea performs the same as RW with c2 ¼ 12 walkers. Let us consider an RW search with K ¼ 12 walkers. At the first hop, the walkers only visit four nodes, but the cost is 12 messages.

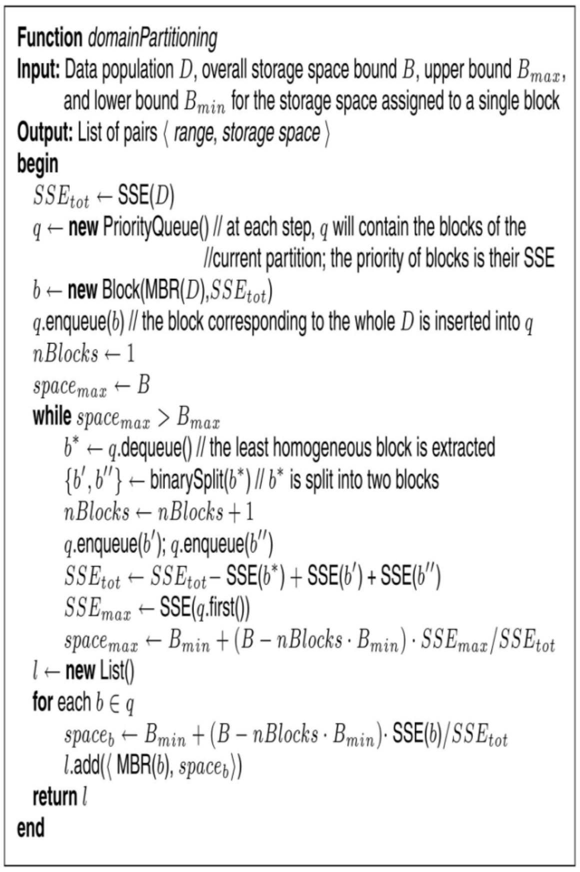
Pseudo code of QuiSea

Algorithm: The pseudo-code of dynamic search QuiSea

```
Input: query source s, queried multidimensional data f, transmission
probability p
Output: the location information of f
QuiSea(s, f, p)
call domain partitioning()
h ← 0
if (h <= n)
h ← h + 1
s choose p portion of its neighbors
```

```
mi carring h visits these chosen neighbors
if f matches index key
return node info
elseif (h > n)
h ← h + 1
mi carring h visits one neighbor of s
foreach (r)
if (r has the location information of f)
r returns the information to s
mi stops
elseif (h > TTL)
mi stops
elseif (h <= n)
h ← h + 1
r choose p portion of its neighbors
mi carring h visits these chosen neighbors
if f matches index key
return node info

elseif (h > n)
h ← h + 1
mi carring h visits one neighbor of r
if f matches index key
return node info
```



```
Function domainPartitioning
Input: Data population $D$, overall storage space bound $B$, upper bound $B_{max}$,
       and lower bound $B_{min}$ for the storage space assigned to a single block
Output: List of pairs ⟨ range, storage space ⟩
begin
  $SSE_{tot} \leftarrow$ SSE($D$)
  $q \leftarrow$ new PriorityQueue() // at each step, $q$ will contain the blocks of the
                                  //current partition; the priority of blocks is their SSE
  $b \leftarrow$ new Block(MBR($D$),$SSE_{tot}$)
  $q$.enqueue($b$) // the block corresponding to the whole $D$ is inserted into $q$
  $nBlocks \leftarrow 1$
  $space_{max} \leftarrow B$
  while $space_{max} > B_{max}$
      $b^* \leftarrow q$.dequeue() // the least homogeneous block is extracted
      $\{b', b''\} \leftarrow$ binarySplit($b^*$) // $b^*$ is split into two blocks
      $nBlocks \leftarrow nBlocks + 1$
      $q$.enqueue($b'$); $q$.enqueue($b''$)
      $SSE_{tot} \leftarrow SSE_{tot} -$ SSE($b^*$) + SSE($b'$) + SSE($b''$)
      $SSE_{max} \leftarrow$ SSE($q$.first())
      $space_{max} \leftarrow B_{min} + (B - nBlocks \cdot B_{min}) \cdot SSE_{max}/SSE_{tot}$
  $l \leftarrow$ new List()
  for each $b \in q$
      $space_b \leftarrow B_{min} + (B - nBlocks \cdot B_{min}) \cdot$ SSE($b$)$/SSE_{tot}$
      $l$.add(⟨ MBR($b$), $space_b$⟩)
  return $l$
end
```

Fig. 3. Data domain partitioning algorithm.

4 Conclusion:

We proposed a framework with compressing, indexing, searching methodologies to simplify searching process. This is applicable to large networks also. In this paper, we have proposed the QuiSea algorithm, which is a generalization of the flooding, MBFS, and RW. QuiSea overcomes the disadvantages of flooding and RW, and takes advantage of various contexts under which each search algorithm performs very well. In our approach, participants make their resources (and possibly their data in a suitable compressed format) available for the other peers in exchange for the possibility of accessing and posing range queries against the data published by others. Our solution is based on suitable data summarization and indexing techniques, and on mechanisms for data distribution and replication that properly take into account the need of preserving the autonomy of peers as well as the interest exhibited by the users in the data to support an efficient query evaluation. The experimental results showed the effectiveness of our approach in providing fast and accurate query answers, and ensuring the robustness that is mandatory in peer-to-peer settings.

References

[1] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger, "Sampling Techniques for Large, Dynamic Graphs," Proc. Ninth IEEE Global Internet Symp. (Global Internet '06), Apr. 2006.

[2] A.H. Rasti, D. Stutzbach, and R. Rejaie, "On the Long-Term Evolution of the Two-Tier Gnutella Overlay," Proc. Ninth IEEE Global Internet Symp. (Global Internet '06), Apr. 2006.

[3] D. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu, "Peer-to-Peer Computing," Technical Report HPL-2002-57, HP, 2002.

[4] K. Sripanidkulchai, The Popularity of Gnutella Queries and Its Implications on Scalability, white paper, Carnegie Mellon Univ., Feb. 2001.

[5] M. Jovanovic, F. Annexstein, and K. Berman, "Scalability Issues in Large Peer-to-Peer Networks: A Case Study of Gnutella," technical report, Laboratory for Networks and Applied Graph Theory, Univ. of Cincinnati, 2001.

[6] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," Proc. 22nd Int'l Conf. Distributed Computing Systems (ICDCS '02), pp. 5-14, July 2002.

[7] G. Kan, "Gnutella," Peer-to-Peer Harnessing the Power of Disruptive Technologies, O'Reilly, pp. 94-122, 2001.

[8] RFC-Gnutella 0.6, http://rfc-gnutella.sourceforge.net /developer/testing/index.html, 2008.

[9] C. Gkantsidis, M. Mihail, and A. Saberi, "Random Walks inPeer-to-Peer Networks," Proc. IEEE INFOCOM '04, pp. 120-130,2004.

[10] L.A. Adamic, R.M. Lukose, A.R. Puniyani, and B.A. Huberman, "Search in Power-Law Networks," Physical Rev., E, vol. 64, 046135,2001.

[11] L.A. Adamic, R.M. Lukose, and B.A. Huberman, "Local Search in Unstructured Networks," Handbook of Graphs and Networks.pp. 295-317, Wiley-VCH, 2003.

[12] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid Search Schemes for Unstructured Peer-to-Peer Networks," Proc. IEEE INFOCOM '05, pp. 1526-1537, 2005.

[13] N. Bisnik and A. Abouzeid, "Modeling and Analysis of Random Walk Search Algorithm in P2P Networks," Proc. Second Int'l Workshop Hot Topics in Peer-to-Peer Systems (HOT-P2P '05), pp. 95-103, 2005.

[14] M.E.J. Newman, S.H. Strogatz, and D.J. Watts, “Random Graphs with Arbitrary Degree Distribution and Their Applications,” Physical Rev., E, vol. 64, 026118, 2001.

[15] H. Wang and T. Lin, “On Efficiency in Searching Networks,” Proc. IEEE INFOCOM ’05, pp. 1490-1501, 2005.

[16] P. Lin, T. Lin, and H. Wang, “Dynamic Search Algorithm in Unstructured Peer-to-Peer Networks,” Proc. Global Telecomm. Conf. (GLOBECOM ’06), Nov. 2006.

[17] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, “Search and Replication in Unstructured Peer-to-Peer Networks,” Proc. 16th Ann. Int’l Conf. Supercomputing (ICS ’02), pp. 84-95, June 2002.

[18] Z. Ge, D.R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, “Modeling Peer-Peer File Sharing Systems,” Proc. IEEE INFOCOM ’03, pp. 2188-2198, 2003.

[19] K. Sripanidkulchai, The Popularity of Gnutella Queries and Its Implications on Scalability, O’Reilly, www.openp2p.com, Feb. 2001.

[20] M. Ripeanu, A. Iamnitchi, and I. Foster, “Mapping the Gnutella Network,” IEEE Internet Computing, vol. 6, no. 1, pp. 50-56, Jan./Feb. 2002.

[21] S. Saroiu, P.K. Gummadi, and .D. Gribble, A Measurement Study of Peer-to-Peer File Sharing Systems. MMCN, Jan. 2002.

[22] J. Chu, K. Labonte, and B. Levine, “Availability and Locality Measurements of Peer-to-Peer File Systems,” ITCom: Scalability and Traffic Control in IP Networks, July 2002.

[23] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, “A Local Search Mechanism for Peer-to-Peer Networks,” Proc. ACM CIKM Int’l Conf. Information and Knowledge Management (CIKM ’02), pp. 300-307, Nov. 2002.

[24] Z. Zhuang, Y. Liu, L. Xiao, and L.M. Ni, “Hybrid Periodical Flooding in Unstructured Peer-to-Peer Networks,” Proc. 32nd Int’l Conf. Parallel Processing (ICPP ’03), pp. 171-178, Oct. 2003.

[25] S. Jiang, L. Guo, and X. Zhang, “LightFlood: An Efficient Flooding Scheme for File Search in Unstructured Peer-to-Peer Systems,” Proc. 32nd Int’l Conf. Parallel Processing (ICPP ’03),
pp. 627-635, Oct. 2003.

[26] S. Jiang, L. Guo, X. Zhang, and H. Wang, “LightFlood: Minimizing Redundant Messages and Maximizing Scope of Peer-to-Peer Search,” IEEE Trans. Parallel and Distributed Systems, vol. 19, no. 5, pp. 601-614, May 2008.

[27] D. Tsoumakos and N. Roussopoulos, “Adaptive Probabilistic Search for Peer-to-Peer Networks,” Proc. Third Int’l Conf. Peer-to-Peer Computing (P2P ’03), pp. 102-109, Sept. 2003.

[28] D. Tsoumakos and N. Roussopoulos, “Analysis and Comparison of P2P Search Methods,” Technical Report CS-TR-4539, UMIACSTR- 2003-107, Dept. of Computer Science, Univ. of Maryland, 2003.

[29] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, “Making Gnutella-Like P2P Systems Scalable,” Proc. ACM SIGCOMM ’03, pp. 407-418, Aug. 2003.

[30] A. Crespo and H. Garcia-Molina, “Routing Indices for Peer-to- Peer Systems,” Proc. 22nd Int’l Conf. Distributed Computing Systems (ICDCS ’02), pp. 23-32, July 2002.

[31] B. Yang and H. Garcia-Molina, “Improving Search in Peer-to-Peer Networks,” Proc. 22nd Int’l Conf. Distributed Computing Systems (ICDCS ’02), pp. 5-14, July 2002.

[32] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, “A Local Search Mechanism for Peer-to-Peer Networks,” Proc. 11th Int’l Conf. Information and Knowledge Management (CIKM ’02), pp. 300-307, Nov. 2002.

[33] R.A. Ferreira, M.K. Ramanathan, A. Awan, A. Grama, and S. Jagannathan, “Search with Probabilistic Guarantees in Unstructured Peer-to-Peer Networks,” Proc. Fifth IEEE Int’l Conf. Peer-to-Peer Computing (P2P ’05), pp. 165-172, Aug. 2005.

[34] N. Sarshar, P.O. Boykin, and V.P. Roychowdhury, “Percolation Search in Power Law Networks: Making Unstructured Peer-to-Peer Networks Scalable,” Proc. Fourth IEEE Int’l Conf. Peer-to-Peer Computing (P2P ’04), pp. 2-9, Aug. 2004.

[35] S. Acharya, V. Poosala, and S. Ramaswamy, “Selectivity Estimation in Spatial Databases,” Proc. 1999 ACM SIGMOD, 1999.

[36] A. Andrzejak and Z. Xu, “Scalable, Efficient Range Queries for Grid Information Services,” Proc. Second Int’l Conf. Peer-to-Peer Computing, 2002.

[37] B. Arai, G. Das, D. Gunopulos, and V. Kalogeraki, “Approximating Aggregation Queries in Peer-to-Peer Networks,” Proc. 22nd Int’l Conf. Data Eng., 2006.

[38] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani, “Estimating Aggregates on a Peer-to-Peer Network,” technical report, Stanford InfoLab, 2003.

[39] N. Bruno, S. Chaudhuri, and L. Gravano, “STHoles: A Multidimensional Workload-Aware Histogram,” Proc. 2001 ACM SIGMOD, 2001.

[40] S. Chaudhuri and U. Dayal, “An Overview of Data Warehousing and OLAP