Supporting Ranked Join in Peer-to-Peer Networks

Keping Zhao[†] Shuigeng Zhou[†] * Kian-Lee Tan[‡] Aoying Zhou[†]

[†]Department of Computer Science & Engineering
Fudan University
Shanghai 200433, China
{kpzhao, sgzhou, ayzhou}@fudan.edu.cn

[‡]School of Computing
National University of Singapore
Singapore 117543
tankl@comp.nus.edu.sg

Abstract

This paper addresses the problem of supporting ranked join in Peer-to-Peer (P2P) networks. Ranked queries produce results that are ordered by a certain computed score. Thanks to the ability of answering information retrieval style queries, ranked query has become a critical need for many applications relying on relational databases, where users do not expect exact answers to their queries, but instead a rank of the objects that best match their preferences. In this paper we propose a novel algorithm $PJoin$ for supporting efficient ranked join queries in P2P networks, which is a part of our work to introduce database query processing facilities in P2P networks. The existing ranked join query algorithms consume an excessive amount of bandwidth when they are applied directly into the scenario of P2P networks. By pruning irrelevant tuples before join probing, PJoin *reduces considerable amount of bandwidth cost. Performance of the proposed algorithm is validated by extensive experiments.*

1. Introduction

Peer-to-Peer (P2P) computing is gaining more and more attention from both academia and industrial community for its scalability, autonomy and fault tolerance. In P2P systems, peers use dedicated naming space and act as both service providers and service consumers, while keeping themselves fully autonomous. Recent P2P systems employ *distributed hash table* (DHT) [14, 13] to overcome performance challenges (e.g. high routing costs) confronted by previous P2P systems like Gnutella[1]. DHT provides theoretical bounds on both the number of routing hops and the number of maintenance messages needed to manage the join and departure of peers in the networks, which is $O(log(n))$ for an overlay network with n peers despite the different implementations. The fundamental limitation of such systems is that they support only *exact-match* queries. Recently, the database community has commenced research work on providing complex query facilities on top of the DHT-based P2P systems. In this paper, we study the problem of *ranked join queries* in the scenario of P2P networks.

Thanks to the ability of answering information retrieval style queries, ranked query has become a vital need for many applications relying on relational databases. Although quite a lot work on ranked query processing has been done in traditional database area, the existing methods can not be applied straightforwardly to P2P context. In traditional databases area, most of the algorithms for answering ranked queries assume that tuples are ordered on some rank attributes. This assumption is reasonable in centralized databases. However, in P2P scenario, because all tuples of a table are stored on different peers in the network, we can not assume the tuples are sorted on some rank attributes. Nor can we sort the tuples before processing, for it is very expensive if not impossible. Moreover, due to the hugeness of P2P network scale and the wide distribution of data, reducing network traffic is one of major concerns of P2P applications, which requires querying processing algorithms to consume bandwidth resource for messages or data exchanging as little as possible

We summarize our contributions in this paper as follows:

1. We propose an efficient algorithm *PJoin* to answer ranked join queries in P2P networks. The algorithm distinguishes itself from previous work on ranked queries over traditional relational databases in many aspects. For example, it makes no assumption that ordered access to the tuples is available. The algorithm can save a considerable amount of bandwidth cost compared with the straightforward methods.

2. Extensive experiments are carried out to evaluate the

*This paper was supported by National Natural Science Foundation of China(NSFC) under grant 60373019, and the Shuguang Scholar Program of Shanghai Municipal Education Committee.

performance of the ranked query algorithms proposed and the impact of different settings on the algorithm are investigated.

The paper is organized as follows. Section 2 surveys the related work. Section 3 presents the problem definitions. Section 4 briefly introduce the $PSel$ algorithm and Section 5 presents the algorithm for ranked join queries . Section 6 gives experimental results. Finally Section 7 concludes the paper and highlights future work.

2. Related Work

There is an emerging set of work which aims to provide complex query facilities and IR-style functionalities in P2P networks. For example, [15] introduces the $pSearch$ information retrieval system, which uses distributed document indices over the P2P network based on document semantics. [8] is closely related to our work, which presents PIER system, a database-style query engine based on DHT overlay networks. Our focus is to develop efficient algorithms to answer specifically ranked join queries in DHT-based P2P networks, which has not been properly addressed in PIER. And [11] proposes an solution based on merging sorting algorithm to answer top-k queries in the HyperCube topology.

There are a number of papers on ranked queries evaluation in the traditional database area. Fagin[4] and Fagin *et al.*[5] introduce the first efficient set of algorithms to answer top-k selection queries in the context of multimedia systems. The database consisting of m attributes is considered as m sorted lists. The TA algorithm[5] assumes the availability of random access to object scores in any list besides the sorted access to each list, while the NRA algorithm[5] makes no random accesses. A set of variants [6, 12] of these algorithms are published later. Bruno *et al.*[3] propose algorithms to evaluate top-k selection queries over web-accessible autonomous databases. They assume that one of the web sources supports ordered access and multiple random-access sources are available.

Ilyas *et al.*[9, 10] propose two algorithms to answer ranked join queries. *NRA-RJ*[10] requires key as the join attributes, thus it supports only the join queries based on key-equality condition. HRNJ[9] is an efficient ranked join query operator based on hash ripple join algorithm, which assumes the inputs ordered individually on the score attributes. In [7], the authors introduce the PREFER system, which answers ranked queries by using materialized views that have been preprocessed and stored.

3. Problem Definitions

In the ranked queries system we consider, the peers share their data in the form of database tuples and relations, where a tuple is stored in the DHT namespace of its corresponding table. We assume that a global schema is known to each peer in the network, and this assumption is also made by some existing systems [8]. We use bandwidth cost as the main performance metric. We argue that bandwidth cost dominates other costs(e.g. cost of computation) in a P2P systems, which is consistent with other published work on P2P systems. Without loss of generality, we assume the domain of the attributes is in $[0, 1]$. Peers are allowed to pose SQL-like ranked queries to the system. The query statement takes the following form, as proposed in [2]:

SELECT **select-list**
FROM **from-list**
WHERE **qualification**
ORDER BY **rank-function**
STOP AFTER **k**

The SELECT, FROM and WHERE clauses are similar to those of the standard SQL query. The ORDER BY defines the *rank function*, which specifies the scores of result tuples. A rank function $\mathcal{F}(x_1, ..., x_n)$ may take attributes of the tables in the from-list as parameters. This paper makes the assumption that the rank function is *monotone*, which is also made by most of the previous work in ranked queries. The last clause, STOP AFTER, sets the number of ordered results that are expected to return by the value of **k**.

We distinguish the *selection* and *join* predicates just as in relational queries, depending on whether the **from-list** consists of one or more tables. For the ranked join queries, we focus on *equi-join* of two relations. In [16], we have proposed solutions for ranked selection queries in P2P networks, which will be briefly introduced in the next section. And in this paper, we mainly address the problem of answering ranked *join* problems.

Definition 1 (Ranked Join Query) *Given two relations R and S, which are stored in the DHT namespace of N_R and N_S respectively, the join attributes $R.j$ and $S.j$, the monotone rank function $\mathcal{F}(r_1, r_2, ..., r_u, s_1, s_2, ..., s_v)$, where r_i (s_i) is an attribute of R (S), and k ($k > 0$),* ranked join query *returns the top k tuples of $R \bowtie_{R.j=S.j} S$, which are ordered by the scores computed by the function $\mathcal{F}$.*

4. The P2P Ranked Selection Algorithm–*PSel*

In this section, we give a brief overview of the algorithm PSel, which is used for solving the selection predicates involved in answering the join queries. For the top k results of the selection query we are seeking, they are stored over *at most* k different peers. If we can limit the query processing to k peers which may have the final results, a large amount

Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05)
1529-4188/05 $20.00 © 2005 **IEEE**

of bandwidth cost will be saved. In order to identify the k nodes, each nodes in the corresponding namespace calculates its local top ranked tuple (i.e. the tuple with the ceiling value of the local scores), and with these ceiling values, the query node then finds out the k nodes which have the top k ceiling values. Then PSel restricts the query processing with the only k nodes to obtain the final results. Interested readers are referred to [16] for more details of the $PSel$ algorithm.

5. The P2P Ranked Join Algorithms

Because the ranked join query involves two relations, and the tuples of the relations are scattered over the overlay network, evaluation of ranked join queries is a nontrivial job. A straightforward strategy may generate the complete join results of $R \bowtie S$, afterwards calculates the top-k join results. However, as shown by previous research only a small fraction of the complete join results is relevant to the final top-k results. The naive strategy suffers from the expensive computation and bandwidth cost on producing the whole join results.

In this section, we present a novel strategy ***PJoin*** for answering ranked join queries in P2P networks. $PJoin$ aims to evaluate the top-k tuples with small bandwidth cost, which is achieved by pruning $irrelevant$ tuples of local nodes before they are sent to be probed for join matches. $PJoin$ is based on the symmetric hash join algorithm in parallel databases. It's straightforward to revise the algorithm to adopt the symmetric semi-join strategy to save more bandwidth cost. In $PJoin$ the DHT facility serves as both the overlay network and the hash tables for storing tuples. A temporary namespace N_J is used to provide the hash-table facilities. To probe the join matches, nodes in N_R and N_S put the $relevant$ tuples of the input tables R and S to the N_J namespace according to the hash key generated on the join attribute. And each node in N_J maintains two hash tables H_R and H_S for R and S respectively, and a priority list L_k with k entries for ranked join results. For each input tuple of R, the responsible node firstly put it in H_R, and scan the proper bucket of H_S to probe the join matches, which will be inserted to the priority list according to their rank value. The procession is similar for the input tuples of S. Different from traditional hash join algorithms and the ones proposed in PIER, only $relevant$ tuples are put into N_J for join probing, and on each node of N_J not all the join matches but the ones with top-k local rank values are maintained and updated. Probing join matches only for the relevant tuples of input tables saves bandwidth and computation cost. Here, the $relevant$ tuple set is the subset of the original input tuple set whose join matches include all the results of the ranked join query.

In order to identify the $relvent$ tuples, the $PJoin$ first calculates a lower bound of top-k results' rank values, then the local tuples whose possible join matches' estimated rank values are above the bound are $relevant$. The details of $PJoin$ algorithm are as follows, which consists of two steps:

1. This step calculates a lower bound on the rank values of the top-k results by sampling method. First, the query node multicasts the query to randomly p nodes in N_R and N_S. These sampled nodes rehash all the local tuples of R table or S table into N_J for join matching. In this way, we have a subset of the whole join results set. Afterwards, the query node calculates the top-k matches from N_J by applying $PSel$ algorithm. Let Γ denote the rank value of the kth join match, which is a lower bound of the rank values of the final results.

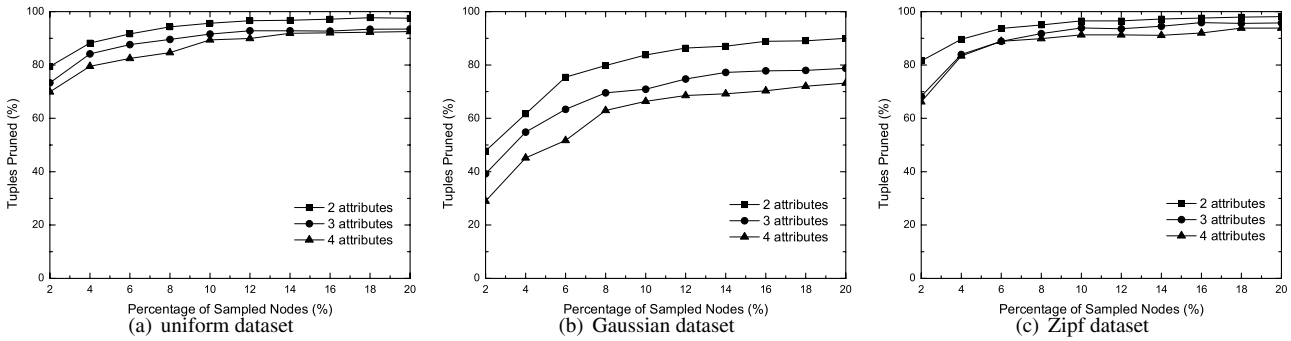
2. The query node multicast the original query along with Γ to the remaining nodes in the namespaces of R and S. When a node in the namespace of R receives the query, it scans the local table of R, and calculates the ceiling value of each tuple's possible join matches. For a tuple r, the ceiling value is computed by the value of $\mathcal{F}(r_1, r_2, ..., r_u, s_1, s_2, ..., s_v)$ where r_i is the corresponding attribute value of r and all s_i are set to 1. For $\mathcal{F}$ is monotone, the value computed is the ceiling value. And the tuples with the ceiling values less than Γ are surely impossible to produce join matches whose rank values are bigger than Γ. So only the tuples whose ceiling value are greater than Γ are $relevant$ and rehashed to the namespace N_J for join probing with the method described above. The peers in the namespace of S also discard the irrelevant tuples which are identified with the similar strategy used by the peers of N_R, only the remaining tuples are rehashed to N_J for join probing. At last, the query node computes the final top-k join results from N_J by making use of $PSel$.

Notice that the tuples hashed to N_J in the first step are still remained in hash tables for probing during the second step. The algorithm ensures that we obtain the right ranked results and saves bandwidth and computation cost. The experimental evaluations show that a big fraction of tuples in practice can be pruned before join probing. Becauseof the limited space, the proof of the correctness of the $PJoin$ algorithm is not covered in this paper, which could be found in [16].

6. Experimental Results

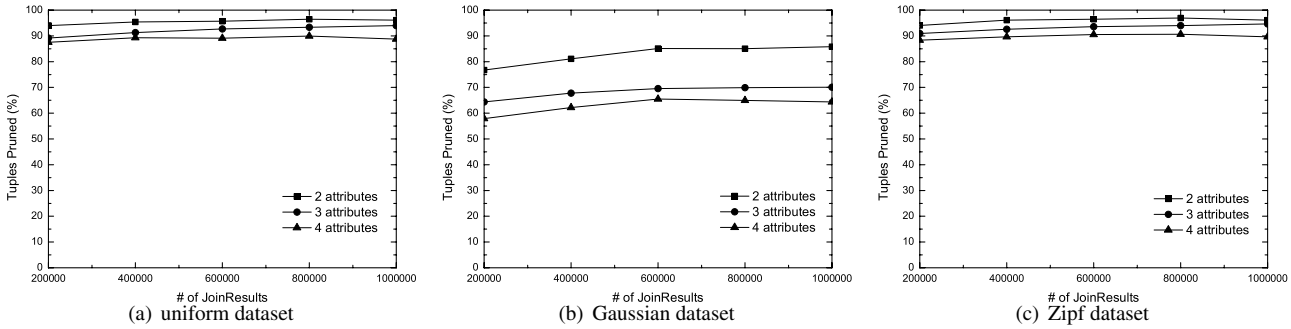
To evaluate the efficiency of the ranked join algorithms proposed herein and the impact of different settings on performance of the algorithms, we carried out a variety of experimental evaluations by simulation. In the scenario of

Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05)
1529-4188/05 $20.00 © 2005 **IEEE**



(a) uniform dataset (b) Gaussian dataset (c) Zipf dataset

Figure 1. *PJoin* **performance** $vs.$ **the number of sampled nodes**



(a) uniform dataset (b) Gaussian dataset (c) Zipf dataset

Figure 2. *PJoin* **performance** $vs.$ **the number of join results**

P2P networks, the performance is greatly impacted by the network bandwidth cost. In the evaluations, we take bandwidth cost as the main performance metric. We first present the simulation and experimental setup, then the experimental results.

6.1. Simulation and Experimental Setup

We implement the algorithms in C++, and run the experiments on a Linux 2.4 workstation with four 2.8G Intel Xeon processors and 2.5G RAM. The implementation makes use of Chord as the DHT facility, and the number of nodes in the overlay network varies from 1000 to 10,000. In order to test the performance of the algorithms over different datasets, we employ 3 kinds of synthetic datasets: uniform dataset, Gaussian dataset and Zipf dataset.

6.2. Evaluating the *PJoin* Algorithm

PJoin aims to prune irrelevant tuples before they are sent to other nodes for join matching. By pruning the irrelevant tuples, the algorithm saves both bandwidth cost and computation cost. And the with PJoin, bandwidth is mainly costed for the tuples' join match probing. Therefore, we use the percentage of tuples pruned before join matching to measure the performance.

The Impact of the number of sampled nodes on performance. Figure 1 shows the performance changing of $PJoin$ when the number of sampled nodes is adjusted. Experiments are conducted over the 3 synthetic datasets, each of which consists of two join tables with 1,000,000 tuples. And the total join results also consist of 1,000,000 matches. We vary the percentage of sampled nodes in the network of 10,000 peers from 1% to 20%, and generate 100 random queries for every setting under which the averaged performance is tested, that is the percentage of tuples pruned in the whole P2P network before join matching. Each figure consists of 3 curves, each of which corresponds to the results of queries with 2-attribute, 3-attribute or 4-attribute rank functions respectively. Figure 1(a) and Figure 1(c) present the results over the uniform dataset and the Zipf dataset respectively. The performance changing patterns over these two datasets are nearly similar. As the number of sampled nodes increases, the percentage of pruned tuples grows. It grows a little fast when the sampling percentage is below 8%; the growing speed slows down after the sampling percentage surpasses 8%. When sampling 5% to 8% percent nodes, over 80% percent of the tuples are pruned. The performance is better for the cases of small rank attributes. Because when the number of rank attributes in the rank function gets bigger, the estimated bound turns looser, and looser bound may remain more irrelevant tuples for join probing.

The impact of the number of total join results on Per-

Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05)
1529-4188/05 $20.00 © 2005 **IEEE**

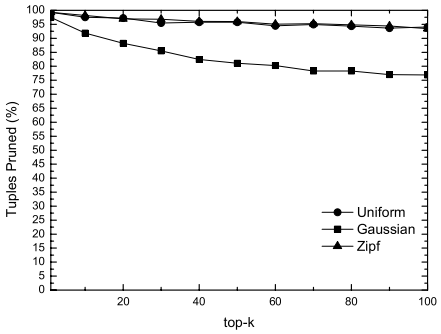


Figure 3. The impact of k on performance of $PJoin$

formance. Figure 2 presents the performance changing of $PJoin$ as the number of total join results varies. The experimental setting is similar to that of the last experiment, except that we fix the sampling percentage to 8%. Experiments are conducted over the datasets that generate different number of join results, which varies from 200,000 to 1,000,000. As the results show, the performance turns better slightly when the number of join results increases. However, the changing is very slow.

The impact of k value on Performance. Figure 3 presents the performance changing of $PJoin$ as the value of k is adjusted over three datasets. The experiments runs over a P2P network of 10,000 nodes, two tables each with 1000000 tuples generate 500000 join results, and 8% nodes are sampled. We increment k from 1 to 100 by the step of 10. The results show as k increases, the performance decreases slightly, and when k is large enough, its impact on performance becomes indistinguishable. We can see that the performance over Gaussian dataset decreases faster than those over the other two datasets. The underlying reason is the same as the number of sampled nodes impacts performance over Gaussian dataset.

7. Conclusions and Future Work

We have presented the new algorithm $PJoin$ for efficiently supporting ranked join queries in P2P networks. The objective of the algorithm proposed herein is to save bandwidth cost when processing ranked queries. The $PJoin$ consists of two steps, which firstly samples the nodes in the P2P network to get a lower bound of the top-k results, then use this bound to prune the irrelevant tuples before join matching. More than 80 percent of tuples can be pruned in our experiments over uniform datasets when 6 percent of nodes are sampled. We plan future work mainly in two directions. In this paper, the algorithms for ranked join queries are based on the symmetric hash join algorithms. It should be valuable to exploit the Bloom join algorithms for answering ranked join queries. Another direction we are interested in is to develop cache mechanism for answering ranked queries in P2P networks.

References

[1] Gnutella Home Page. *http://www.gnutella.com/*.
[2] M. J. Carey and D. Kossmann. On saying "enough already!" in sql. In *SIGMOD '97*, 1997.
[3] S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *VLDB '99*, 1999.
[4] R. Fagin. Combining fuzzy information from multiple systems (extended abstract). In *PODS '96*, 1996.
[5] R. Fagin, A. Lote, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS '01*, 2001.
[6] U. Guntzer, W.-T. Balke, and W. Kiebling. Towards effiecient multi-feature queries for image databases. In *VLDB '00*, 2000.
[7] V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: a system for the efficient execution of multi-parametric ranked queries. In *SIGMOD '01*, 2001.
[8] R. Huebsch, J. Hellerstein, N. Lanham, B. Loo, and S. Shenker. Querying the internet with PIER. In *VLDB '03*, 2003.
[9] I. Ilyas, W. Aref, and A. Elmagarmid. Joining ranked inputs in practice. In *VLDB '02*, 2002.
[10] I. Ilyas, W. Aref, and A. Elmagarmid. Supporting top-k join queries in relational databases. In *VLDB '03*, 2003.
[11] W. Nejdl, W. Siberski, U. Thaden, and W.-T. Balke. Top-k query evaluation for schema-based peer-to-peer networks. In *ICDE 2005*, 2005.
[12] S. Nepal and M. V. Ramakrishna. Query processing issues in image(multimedia) databases. In *ICDE '99*, 1999.
[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01*, 2001.
[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01*, 2001.
[15] C. Tang, Z. Xu, and M. Mahalingam. Psearch: Information retrieval in structured overlays. *HotNets-I*, 2002.
[16] K. Zhao, S. Zhou, and A. Zhou. Supporting ranked queries in peer-to-peer networks. Technical report, Fudan University, May 2004.

Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA'05)
1529-4188/05 $20.00 © 2005 **IEEE**