Journal of Theoretical and Applied Information Technology
31st March 2013. Vol. 49 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

EFFICIENT TOP-K QUERY PROCESSING IN P2P NETWORKS

[1]HUI WANG, [2] ZHITAO GUAN, [3] YUE XU

[1]School of Economy and Management, North China Electric Power University, Beijing

[2,3]School of Control and Computer Engineering, North China Electric Power University, Beijing

E-mail: [1]wanghuibd@ncepu.edu.cn, [2]guan@ncepu.edu.cn

ABSTRACT

Top-k query is widely used in the search engine and gains great success, which perform the cooperative query by aggregating the database objects' degree of match for each different query predicate and returning the best k matching objects only. It's also applied to p2p file-sharing systems. However, top-k query processing in p2p systems is very challenging because the potentially large number of peers may contribute to the query results, which may induce a huge amount of network traffic and high latency. In this paper, we develop a framework that can effectively satisfy the demands. Two algorithms are used for local peer query processing and hierarchical join query processing, and two schemes are proposed to deal with the problem of peer's dynamicity and further reduce communication cost. Simulation results show that our algorithms and schemes are effective.

Keywords: *Peer-to-peer (P2P), Top-k query, algorithm*

1. INTRODUCTION

Peer-to-peer (P2P) networks such as Gnutella, KaZaA, and BitTorrent have emerged as a new Internet computing paradigm over the past few years. However, p2p applications are limited because only keyword-based query is supported currently. Top-k query scheme is widely applied in search engines and gains great success. In some famous search engines such as google and baidu, for each query, there will be top 10 most matched web pages returned. We hope such scheme can also be applied to p2p applications. That is, after the querying peer sends out a query, top k most matched objects can be returned. The user usually is only interested in a small percent of the matched objects instead of all matched ones in network. Applying top-k query in p2p system will be very attractive since the network bandwidth will be saved, query response time will be shortened and the user will also be more satisfied.

Section 2 presents the related work. In section 3, we propose our top-k query model and algorithms. Section 4 shows the simulation results. Section 5 gives a conclusion to the whole paper.

2. RELATED WORK

Top-k query processing has received much attention in a variety of settings such as similarity search on multimedia data[1, 2], ranked retrieval on text and semi-structured objects in digital libraries and on the Web[3, 4, 5], network and stream monitoring [6, 7, 8,9], and ranking of SQL-style query results on structured data sources in general [10, 11,12,13,14,15,16].

In terms of efficiency, the most widely recognized algorithm for top-k queries in a centralized environment is the Threshold Algorithm (TA) [17]. TA starts out by performing a parallel sorted access to the k lists. While an object o_i is seen by the query peer, TA performs a random access to the other lists to find the exact score for o_i (i.e. $\sum_{j=1}^{n} o_{ij}$). After finding the exact score for each object in the current row, it computes a threshold value τ as the sum of all exact scores in the current row. The algorithm stops after k objects have been found with a score above τ . While the TA algorithm uses many round trips as it invokes several small random accesses. This would again translate into an arbitrary large number of phases, which is highly undesirable for a hierarchical environment.

While TA algorithm is not fit for the distributed hierarchical environment for it may cause great latency. So, many other top-k algorithms have been proposed. Bruno [18] discusses the problem of answering top-k queries over web accessible

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

databases. The problem of continually providing top k answers in a distributed environment is discussed in Babcock's work [19]. The problem is tackled by installing arithmetic constraints at each peer which define the current top-k scores at any point. The problem was later extended to hierarchical environments [20]. The TPUT [9] algorithm proposed by Cao and Wang, uses three phases in order to resolve top-k queries in star topologies. The algorithm constructs a bound which is uniform for all lists, similarly to FA, which is too coarse in practice. Finally the recent work [21] examines the problem of approximate top-k queries in distributed environments. The paper assumes that each peer maintains an approximation of the local scores instead of the actual scores. The approximation essentially consists of an equi-width histogram on the local scores along with a bloom filter per histogram bucket which captures object identifiers inserted into the specific bucket.

However, most of these techniques are fairly well understood for centralized data management, but much less explored for distributed systems such as p2p or sensor networks. For example, building a p2p Web search engine where thousands of nodes collaborate to provide Google functionality in a decentralized and self-organizing manner would be a great application for distributed top-k query processing. And in this paper, we present HPJT, a p2p top-k query processing algorithm to tackle this problem.

3. HPJT

3.1. Query model

It is impossible to model the complete dynamics of a p2p system. While our simple models do not capture all aspects of reality, we only focus on two most important aspects: (1) Network topology; (2) Definitions about top-k query in p2p network. We hope the model capture the essential features needed to understand the fundamental qualitative differences between our algorithms and other ones.

3.1.1. Network topology

Gnutella is one of the most popular p2p applications, so we use the gnutella-like topology in our study. For the latest two-tier Gnutella network, Stutzbach [22] provided a detailed characterization of p2p overlay topologies and their dynamics. As shown in figure 1, a small subset of peers becomes super peers, be responsible for the management of search process for their leaves and forward query packets from other super peers. super peers are averagely connected to 30 other peers, while leaf peers hold only a small number of connections to super peers. Leaf peers, on the other hand, concentrate on providing files. Information about the files the peers share is uploaded to the super peer. The average peer connectivity degree of modern gnutella is about 30, which is far greater than the former result 3.1[23]. Power-law distribution is no longer fit for the modern gnutella which shows the characteristic of the dense p2p network. In this hierarchical network, the query of the leaf peer firstly is sent to a super peer, then the super peer forwards the query according to the predefined rules, finally the super peer collects the results and returns them to the leaf peer. Hereafter, a peer refers to a super peer.

Different from former research, we won't assume the topology do not change during the simulation of our algorithms, but take peer's dynamicity into consideration.

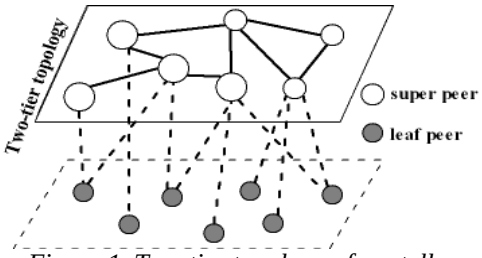


Figure 1. Two-tier topology of gnutella

3.1.2 Definitions about top-k query.

Below is a SQL-like template for expressing top-k aggregate queries and an example query Q.

TEMPLATE	EXAMPLE
SELECT *DATA*, *F*	SELECT O_1, O_2,…,O_N, AGGREGATE(O_I)
FROM R_1, ...,R_N	AS SCORE
GROUP BY *DATA*	FROM P_1, P_2, …,P_M GROUP BY O_1, O_2, …, O_N
ORDER BY *F*	ORDER BY SCORE
LIMIT *K*	LIMIT K

That is, the groups are ordered by a ranking aggregate *F*, and the top k groups with the highest *F* values are returned as the query result. For simplicity, we assume the sum function as the ranking aggregate function in this paper. Now we give some basic definitions.

Definition 1 Aggregate Function *F*

For a queried object o_i, whose aggregation function is

$$F[o_i] = F(o_i^1, o_i^2, ..., o_i^m) = \sum_{j=1}^{m} w_j * score(o_i^j) \quad (1)$$

Journal of Theoretical and Applied Information Technology
31st March 2013. Vol. 49 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

where w_j is the weighted factor and we assuming it is 1 here. $score(o_i^j)$ is the score of o_i in peer j (p_j) for the given query. Usually the query and queried objects (text object, multimedia etc.) can be expressed as n-attribute vector. Consider one top-k query $q=(q_1,q_2,...,q_n)$, the object (document) $o_i=\{o_{i1},o_{i2},...,o_{in}\}$, $score(o_i^j)$ can be given as following:

$$score(o_i^j)=sim(q,o_i^j)=\vec{q}\cdot\vec{o}_i^x \Big/ |\vec{q}|\times|\vec{o}_i^x| \quad (2)$$

where $sim(q,o_i^j)$ is the similarity function which evaluates the j[th] attribute of the query q against the j[th] attribute of the object o_i and returns a value in the domain [0,1] (1 denotes the highest similarity). Note that, similarly to research [9, 16], we require the score function to be monotone. A function is monotone if the following property holds:

If $sim(q_i,o_{1i})>sim(q_i,o_{2i})\ (\forall i\in n)$, then $F(o_1)>F(o_2)$. This is true when $\theta_j>0\ (\forall j\in n)$. Obviously, from formula (1) and (2), our aggregation function satisfies such acquirement.

If the local scores of all objects can be collected by the querying peer, it will be quite simple to compute the exact aggregate score F[o_i] for each object o_i, and then top-k objects can be got directly by sorting F[o_i] in descend order. However, it will induce heavy network traffic. A better approach is to make some estimation and only compute the score of a few most possible candidate objects. Thus, the partial aggregate function and upper-bound aggregate function are introduced.

Definition 2 Partial Aggregate Function F

The partial aggregation function of o_i is

$$F_{Partial}[o_i]=\sum_{j=1}^{m} score_{Partial}(o_i^j), \quad (3)$$

$$score_{Partial}(o_i^j)=\begin{cases} score(o_i^j), if\ o_i\ be\ seen\ in\ node\ j \\ 0, otherwise \end{cases}$$

Definition 3 Upper-bound Aggregate Function F

The upper-bound aggregation function of o_i is

$$F_{Upper}[o_i]=\sum_{j=1}^{m} score_{Upper}(o_i^j), \quad (4)$$

$$score_{Upper}(o_i^j)=\begin{cases} score(o_i^j), if\ o_i\ be\ seen\ in\ node\ j \\ \delta(j), otherwise \end{cases}$$

$\delta(j)$ is the score of the last seen object in sorted access in peer j.

From the above definitions and statement, for any object o, the following inequation stands.

$$F_{Partial}[o_i]\le F[o_i]\le F_{Upper}[o_i] \quad (5)$$

The algorithm proposed in this paper will make use of inequation (5) as basis.

3.2. Dynamic query tree

```
1. PROCEDURE GENERATEQUERYTREE(PEER P)
2.   IF P RECEIVES QUERY FROM PEER Q
3.     IF P RECEIVES THE QUERY FOR THE FIRST TIME
4.       MARK Q AS PARENT PEER;
5.     ELSE
6.       SEND DUPLICATED MESSAGE SIGNAL TO P;
7.     END IF;
8.   END IF;
9.   LOOP FOR EACH NEIGHBOR X OF P
10.    SEND QUERY TO X, AND MARKS X AS ITS CHILD
       PEER;
11.  END LOOP;
12.  IF P RECEIVES DUPLICATED SIGNAL FROM
       NEIGHBOR Y
13.    REMOVE Y FROM P'S LEAF PEER SET.
14.  END IF;
15. END PROCEDURE
```

In unstructured p2p networks, query is typically processed as follows. The query is sent out from the initiator to its neighbors until the TTL (Time-to-live) value of the query decreases to 0 or the current peer has no peer to forward. The query won't be forwarded when TTL is 0. The query must be single-direction to avoid query loop. So the query processing flow can be represented as a tree, which is called as dynamic query tree.

Procedure GenerateQueryTree will be executed in each peer the query passed through. The peer firstly marks the parent peer (line 2-8), and then marks children peers (line 9-14).

3.3. Basic algorithm

3.3.1. Algorithm statement

Different from the existed top-k query algorithms, one key feature of our algorithm is performing computation along the query path instead of only centralized processing in the root. The algorithm comprises the following four steps.

Step 1. Get local top-k objects.

Initialize the query q, and set TTL. According to procedure GenerateQueryTree, the dynamic query tree with TTL layers is firstly generated, and the root of the tree is the querying peer. Each peer in query tree gets local top-k objects of which scores are highest, and the score of the object is got according to formula (2). Then it returns its local top-k objects list to its parent peer. The parent peer merges all results from its children peers and its own top-k results into a new set, as shown in

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

formula (6), then forward the set to the upper layer. Such merge-and-backward process executes recursively until the querying peer gets the final result sets List_root. In order to minimize network traffic, we do not bubble up the top object items (which could be large), only their local scores and addresses. A returned list is simply a list of k couples (*obj*, *local_score*, *addr*), such that *local_score* is the score of *obj* in the local peer and *addr* is the address of the peer owning the object item.

$$List(p) = local\_list_k(p) \cup \left(\bigcup_{\forall x \in children(p)} local\_list_k(x) \right) \quad (6)$$

Step 2. Estimate the lower bound.

According to the final list List_root, the query peer calculates the partial aggregate score of the included objects according to formula (3). It sorts the objects by partial aggregate score in descending order and gets kth score as tempKscore. The lower bound K_{TH} is got as: K_{TH} = tempKscore/n, where n is the total number of queried peers.

Step 3. Hierarchical join query.

The root peer packs the two parameters List_root and K_{TH} to a new message and spread it to all peers in query tree. After receives the message, each peer unpacks it and executes the operation of selecting local candidate objects according to the received two parameters. This process is detailed stated in the following procedure.

```
1. PROCEDURE GETLOCALLIST(PEER P)
2. LOOP FOR EACH OBJECT O IN LOCAL LIST OF
     PEER P
3.   IF (O.SCORE>K_TH) LIST1.ADD(O);
4. END LOOP;
5. LOWESTRANK:=0;
6. LOOP FOR EACH OBJECT O IN LIST_ROOT_1ST
7.   IF(LOWESTRANK < O.LOCAL_RANK)
8.     LOWESTRANK = O.LOCAL_RANK;
9.   END IF;
10. END LOOP;
11. LOOP I=1 TO LOWESTRANK
12.   LIST2.ADD(LOCAL_SORTED_LIST(I));
13. END LOOP;
14. LOCAL_LIST=LIST1 ∩ LIST2;
15. RETURN LOCAL_LIST;
16. END PROCEDURE
```

The procedure GetLocalList will be executed in each peer. It firstly gets the set list1 which is comprised of the objects whose local score is higher than K_{TH}. Then finds the min rank of the objects in List_root locally, and gets all local objects whose ranks are not lower than the min rank as the set list2. The local candidate list is got as $local\_list = list1 \cap list2$. The result list got from the procedure GetLocalList will be called in the procedure for the hierarchical computation.

```
1. PROCEDURE HIERARCHICALCOMPUTE (PEER P)
2. DEFINE LIST UP_LIST<OBJ, SCORE, UB_PEERS,
     UB_FLAG>
     //UB_FLAG: BOOL PARAMETER, IF SCORE IS
                UPPER BOUND,
     //IT'S TRUE, ELSE IT'S FALSE. IT'S DEFAULT
                VALUE IS FALSE;
3. LET UP_LIST = ∅;
4. UP_LIST = GETLOCALLIST(P);
5. IF PEER P IS NOT LEAF PEER
6.   PEER PEERS[] = P ∪(children peers of p);
7.   Object objs[] = obj in local_list
                ∪( ⋃_{∀x∈children(p)} obj in up_list(x) );
8.   DEFINE NUMBER AGG_SCORE[],
                LIST<PEER> UB_PEERS[], BOOL
                UB_FLAG[];
9.   LOOP I=1 TO SIZE OF OBJS[]
10.    LOOP J=1 TO SIZE OF PEERS[]
11.      IF OBJ[I] IS IN UP_LIST[J]( UP_LIST OF
       PEERS[J] )
12.        AGG_SCORE[I] =
                AGG_SCORE[I]+UP_LIST[J
                ].SCORE;
13.      ELSE
14.        UB_SCORE = MIN(SCORE IN UP_LIST[J]);
15.        AGG_SCORE[I] =
       AGG_SCORE[I]+UB_SCORE;
16.        UB_FLAG = TRUE;
17.        ADD PEER[J] TO LIST UB_PEERS[I];
18.      END IF;
19.      IF UP_LIST[J].UB_FLAG IS TRUE
20.        UB_FLAG = TRUE;
21.      END IF;
22.    END LOOP;
23.  END LOOP;
24. END IF;
25. LOOP M =1 TO SIZE OF OBJS[]
26.   ADD < OBJS[M], AGG_SCORE[M],
                UB_PEERS[M], UB_FLAG[M] > TO
                UP_LIST;
27. END LOOP;
28. RETURN UP_LIST;
29. END PROCEDURE
```

If peer p is a leaf peer, the aggregate score of the objects in up_list (the result list to upload to its parent peer) is only the local score.

Else if peer p has some children peers, the approach to get up_list will take some sub-steps.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Firstly, all objects are collected from the up_lists of peer p and its children peers without duplication (line 6). Secondly, the aggregate score of the collected objects is computed (line 8 - line23). For each object *obj*, if it has been found in up_list of all peers, its exact (partial) aggregate score can be got by adding up its score in all lists directly; else if it isn't found in some peer, say M, its score in M can be estimated by using upper bound score, which is the min score in the ub_list of M. Then the upper bound aggregate score of *obj* can be got according to formula (4) in section 3.1.2. We use a Boolean flag parameter ub_flag to mark the score in up_list is exact score or upper bound score, as shown in line 15, the object with upper bound score will be marked. And, the address of the peer using upper estimated score will be added to the list ub_peers (line 16), which will be convenient for the last step to verify the true top-k results.

Step 4. Clean-up process.

The querying peer has collected a list of objects for which either the complete aggregate score or an upper bound aggregate score has been computed. It sorts the objects in the result list of step 3 by their complete score or upper bound in descending order. Then

✧ **If** all the scores of the top-k objects in final list List_root_2nd are complete aggregate score, then get the top-k results directly.

✧ **Else** the querying peer finds those objects that have upper bounds higher than the k-th complete score and computes the exact scores for these by requesting the exact scores from its children. Then get the true top-k results.

3.3.2. Correctness analysis of the algorithm

The algorithm includes two main query round-trips, that is, step 1 and step 3. It firstly gets top-k local objects from all queried peers in step 1. And in step 2, two important parameters, the lower bound K_{TH} and list_root are got for the next query round-trip. In the second query round-trip, that is, in step 3, K_{TH} and list_root are spread out to all queried peers. Each executes procedure GetLocalList according the received two parameters. The returned list in each peer undergoes merge-and-backward process by the procedure HierarchicalCompute recursively, finally to the querying peer.

At the end of the second round-trip (step 3), the querying peer has seen objects in the true top-k set. The reason is that the procedure GetLocalList can collect the true top-k candidates from queried peers. The analysis is as following.

✧ For *list1*. If one object whose score is < K_{TH} in all peers, which means that its aggregate score is < tempKscore, and hence it can't be in the top-k set. Then the true top-k objects must be included in *list1* of one peer at least.

✧ For *list2*. Each peer receiving *list_root*, searches its local sorted list in order to identify the index of the lowest ranked object that belongs to *list_root*. All objects above *lowestRank* are added to *list2*. Obviously the true top-k objects are also included in *list2* of one peer at least.

Then the true top-k objects must be included in the list *local_list* of one peer at least, which is returned by procedure GetLocalList, since it is the intersection of *list1* and *list2*. The final object list in querying peer is the combination of the *local_list* in each peer, hence after the step 3, the querying peer has seen objects in the true top-k set.

In the last step, the querying peer identifies the top-k objects.

3.4. Enhancing the power of HPJT under failures

P2P network a dynamic environment in which peers appear to be leaving or joining the network in an ad-hoc manner. This might not allow the querying node to obtain the correct top-k results during its execution. And the connections between peers are also unstable, which may induce unpredictable delay in query processing.

3.4.1. Problems from peers' dynamicity

Execution of top-k query algorithm in p2p network may encounter the following problems.

1. In case that the message is transferred in top down sequence along the query tree. Say there's peer x, whose child peer y leave the network in transferring process, which could result in that all children peers of peer can't get the message.

2. In case that the message is transferred in bottom up scheme. Say peer x is the parent peer of peer y.

(a). Peer x leave the network dynamically, which induce the merged information in peer y can't be uploaded.

(b). Late reception of feedback message from children peers. In p2p network, usually each peer will set a wait time threshold, and after its wait time has expired, it will merge its local top list with the lists received from its children and send the result

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

list to its parent. However, a peer may underestimate its wait time which is based, among other parameters, on local processing parameters of other peers. Thus, it may happen that lists of some children arrive late, i.e. after it has sent its list to its parent.

3.4.2. Solution strategies

1. Strategy 1: Backup-parent peers. When generating query tree, each peer will keep a backup-parent peer.

✧ Problem 1 may happen in step 3, that is, in case that the root spread K_{TH} and list_root to all peers. After step 1, each peer will set its wait time threshold. When the wait time expires, it will send a test packet to its parent to verify whether the parent is online. If its parent is offline, it will send a *change_parent* message to its backup-parent peer. And then get message from its new parent.

✧ The problem 2.(a) may happen in step 1 and step 3 of our algorithm. We can make a small change to our algorithm by adding a new rule. In bottom up scheme, each parent should give back responses to its children peers after it receive their packets. So, if the wait time of the child peer expires, it can verify the connection to the parent to decide whether to change parent and upload information to its backup-parent peer. From the discussion about algorithm *HierachicalCompute*, it's obvious that *change_parent* operation won't affect the correctness of the algorithm.

2. Strategy 2: additional list. The problem 2.(b) may happen in step 1 and step 3. It can be solved by following strategy. When a peer, say p, receives the late message list from some child peer. It will mark it as additional list. The additional list should be bubbled up without wait until arriving to a peer, say q, of which wait time has not expired. And peer q deals with the additional list as any other received list.

4. PERFORMANCE EVALUATION

4.1 Environment setup

We experimentally evaluated the performance of our proposed algorithm HPJT and related algorithms in p2p networks. The topology and data settings are set as following.

⑴ Topology setting.

Topologies in our simulation are generated by BRITE [24], a universal topology generator. To verify the scalability of HPJT algorithm, 4 kinds of topologies are used, whose size are 1024, 2048, 4096 and 10240 respectively.

⑵ Data setting.

The data sets for simulation were generated for performance evaluation as follows. Assume each peer has n objects. We assumed that peers exhibit different degrees of correlation of each other. Firstly n values $s_1,...,s_n$, which follow the Zipf's distribution [23] with a Zipf factor α, are generated. These n values are assigned to the n objects as their scores in peer 1. Then, a random walk model was used to generate the scores of the objects in other peers. For peer i, $Score[i] = Score[i-1] + Var_i$, where $Score[i]$ is the score of object O at peer i and Var_i is a random number in the range $[-c \times Score[0], +c \times Score[0]]$, c is a constant. By varying α and c, different scenarios such as the scenario in which the object rankings are similar in different peers or the scenario in which the object rankings vary in different peers can be generated. For simplicity, we set c=0.1, $\alpha = 0.3$.

We compared the performance of our new HPJT algorithm with the following two techniques.

✧ C-TPUT: This is an efficient 3-phase algorithm for distributed networks as described in [9].

✧ H-TPUT: This is a varied TPUT algorithm for more efficient top-k query processing in hierarchical distributed networks such as p2p networks [9].

Our implementation of the test-bed was written in java. We take response time and bandwidth cost as metrics and measured in two conditions:

⑴ Topology varies.

This is used to verify the scalability of the algorithms. We generate 4 kinds of topologies. The number of the peers is 1024, 2048, 4096 and 10240 respectively. *K* is set as 10 in this condition, that is, queries in this condition were for the top-10 results.

⑵ Parameter *K* varies.

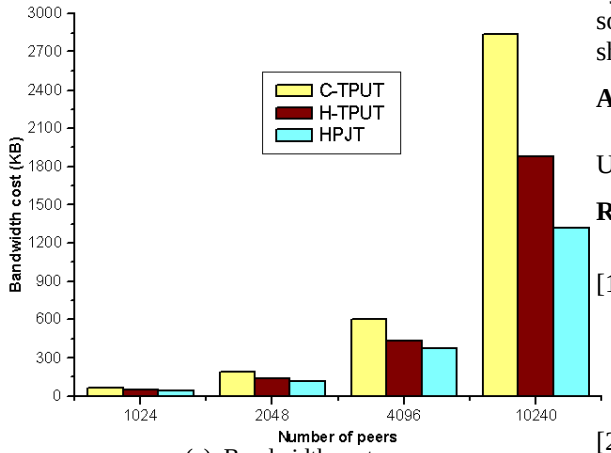
This is used to verify the performance of the algorithms with different number of results needed. We choose eleven *K* values from [1,100] uniformly. Topology size used for this condition is 4096.

4.2 Results and analysis

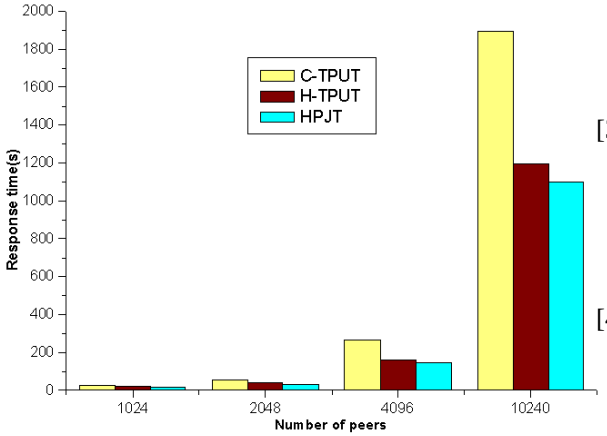
Figure 2(a) shows the bandwidth cost of producing the top-10 results in different topology scales. It's seen HPJT algorithm shows better

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

performance than the two compared algorithms. H-TPUT and HPJT are better than C-TPUT. This is because H-TPUT and HPJT prune some results in query backward phase along the query path, while C-TPUT collects all query results back to the query peer without pruning. The main reason that HPJT consumes less bandwidth than H-TPUT is HPJT prunes more potentially useless objects in hierarchical join query phase (See step 3 in section 3.3.1).



(a) Bandwidth cost



(b) Response time

Figure 2. Performance comparison in different topology scales

Figure 2(b) shows the response time with the setting in figure 2(a). HPJT and H-TPUT have similar results and both of them are better than C-TPUT. There are two main reasons. Firstly, in C-TPUT algorithm, the computation is only performed in query peer, while the other two algorithms "put computation to the network". Secondly, similar to the explanation to figure 2(a), C-TPUT collects all local results in queried peers without pruning. This also will cause more delay. The reason that HPJT is better than H-TPUT, we think it mainly got that HPJT prunes more objects and then decreases the delay.

5. CONCLUSION

In this paper, an efficient p2p top-k query processing scheme named HPJT is presented. A fully distributed query processing algorithm for top-k query is proposed, which performs computation along the query path instead of only centralized processing in the root. And the problem of peer's dynamicity is tackled by some simple but effective solutions. The simulation results indicate HPJT shows good performance in p2p environment.

ACKNOWLEDGEMENT:

This work was supported by Central Government University Foundation (Grant No. JB2012087).

REFRENCES:

[1] S. Chaudhuri, L. Gravano, A. Marian, "Optimizing Top-K Selection Queries over Multimedia Repositories", *IEEE transactions on knowledge and data engine* , Vol. 16, No. 8, 2004, pp. 992–1009.

[2] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. "Monochromatic and bichromatic reverse top-k queries". *IEEE transactions on knowledge and data engine* , Vol. 23, No. 8, 2011, pp. 1215–1229.

[3] M. A. Soliman, I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi. "Ranking with uncertain scoring functions: semantics and sensitivity measures". *Proceedings of SIGMOD 2011*, Athens, Greece, June 12-16, 2011, pp. 805–816.

[4] M. Theobald, G. Weikum, R. Schenkel. "Top-k Query Evaluation with Probabilistic Guarantees". *Proceedings of VLDB 2004*, Toronto, Canada, August 29-September 3, 2004, pp.648-659.

[5] R. Kaushik, et al. "On the Integration of Structure Indexes and Inverted Lists". *Proceedings of SIGMOD Conference 2004*, Paris, France, June 3-8, 2004, pp.779-790.

[6] M. Wu, J. Xu, X. Tang and WC. Lee, "Top-k Monitoring in Wireless Sensor Networks". *IEEE transactions on knowledge and data engine* , Vol. 19, No. 1, 2007, pp. 962-976.

[7] B. Babcock, C. Olston. "Distributed Top-K Monitoring". *Proceedings of SIGMOD Conference 2003*. San Diego, CA, USA, 2003, pp. 28-39.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[8] N. Koudas, et al. "Approximate NN queries on Streams with Guaranteed Error/performance Bounds". *Proceedings of VLDB 2004*, Toronto, Canada, August 29-September 3, 2004, pp.804-815.

[9] P. Cao, Z. Wang. "Efficient Top-K Query Calculation in Distributed Networks". *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, Newfoundland, Canada, July 25-28, 2004,pp.206-215.

[10] A Vlachou, C Doulkeridis, K Nørvåg, M Vazirgiannis. "On Efficient Top-k Query Processing in Highly Distributed Environments", SIGMOD Conference 2008.

[11] R. Akbarinia, V. Martins, E. Pacitti, P. Valduriez. "Top-k Query Processing in the APPA P2P System". *Proceedings of LNCS*, Rio de Janeiro, Brazil, June 10-13, 2007, pp. 158-171.

[12] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput-ing Surveys*, Vol.40, No 4, 2008, pp.1-58.

[13] S. Chaudhuri, et al. "Probabilistic Ranking of Database Query Results". *Proceedings of VLDB 2004*, Toronto, Canada, August 29-September 3, 2004, pp.648-659.

[14] N. Bruno, S. Chaudhuri, L. Gravano. "Top-k selection queries over relational databases: Mapping strategies and performance evaluation". *Proceedings of TODS*, 27(2), 2002

[15] QH. Vu, BC. Ooi, D. Papadias, AKH. Tung. "A Graph Method for Keyword-based Selection of the top-K Databases". *SIGMOD Conference 2008*. , Vancouver, Canada, June 9-12, 2008 , pp.915-926.

[16] A. Vlachou, C. Doulkeridis, K. Norvag, M. Vazirgiannis. "Skyline-based Peer-to-Peer Top-k Query Processing". *Proceedings of ICDE 2008*. Cancun, Mexico, April 7-12, 2008, pp. 1421-1423.

[17] R. Fagin, J. Lotem, M. Naor. "Optimal aggregation algorithms for middleware". *Journal of Computer and System Sciences* Vol.66, No.4, 2003, pp. 102-113.

[18] N. Bruno, S. Chaudhuri, L. Gravano. "Top-k selection queries over relational databases: Mapping strategies and performance evaluation". *ACM Transactions on Database Systems*, Vol.27, No. 2, 2002, pp.153-187.

[19] B. Babcock and C. Olston, "Distributed Top-K Monitoring", *Proceedings of the ACM SIGMOD international conference on Management of data*, San Diego, CA, USA, 2003, pp.28-39.

[20] A. Deligiannakis, Y. Kotidis, N. Roussopoulos "Hierarchical in-Network Data Aggregation with Quality Guarantees", *9th International Conference on Extending Database Technology*, Heraklion, Greece, March 14-18,2004, pp. 658-675

[21] S. Michel, P. Triantafillou, G. Weikum "KLEE: A Framework for Distributed Top-k Query Algorithms", *31st conference in the series of the Very Large Databases*, Trondheim, Norway, 2005, pp..637-648

[22] D. Stutzbach, R. Rejaie, S. Sen. "Characterizing unstructured overlay topologies in modern p2p file-sharing systems", *Proceedings of Internet Measurement Conference*. Berkeley, CA, USA, October 19-21 2005, pp. 49-62.

[23] B. Yang, H. Garcia-Molina. "Designing a Super- peer Network", *Proceedings of IEEE International Conference on Data Engineering*. Bangalore, India, March 5-8, 2003, pp. 49-60.

[24] A Medina, A Lakhina, I Matta, J Byers: BRITE: an approach to universal topology generation. *Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. Cincinnati, OH, Aug 15-18,2001, pp. 346-353.