Evaluation of Top-k queries in Peer-to-Peer Networks using Threshold Algorithms

Ioannis Chrysakis
ICS-FORTH
N. Plastira 100
Vasilika Vouton GR-70013
+30 2810391683
hrysakis@ics.forth.gr

Constantinos Chalkidis
ICS-FORTH
N. Plastira 100
Vasilika Vouton GR-70013
+30 2810391683
conhalk@csd.uoc.gr

Dimitris Plexousakis
ICS-FORTH
N. Plastira 100
Vasilika Vouton GR-70013
+30 2810391637
dp@ics.forth.gr

ABSTRACT
In p2p networks, top-k query processing can provide a lot of advantages both in time and bandwidth consumption. Several algorithms have been proposed for the evaluation of top-k queries. A large percentage of them follow the Threshold Approach. We focus on the main adaptations of threshold algorithms fulfilling the requirements of modern p2p applications. We introduce two algorithms optimized for ranking queries in p2p networks and present their characteristics. In the setting of a simulation of large p2p networks, we evaluate the algorithms’ performance. Our experiments demonstrate that in some cases a threshold algorithm can improve top-k query processing, while in others it is far more costly. The results show that distributed query processing can be more effective than a simple threshold algorithm in a p2p network.

Categories and Subject Descriptors
H.2.4 [Systems]: Distributed databases. Query processing

General Terms
Algorithms, Experimentation, Measurement, Performance

Keywords
Top-k queries, query processing, peer-to-peer networks

1. INTRODUCTION
Ranking queries, also known as top-k queries, produce results that are ordered on some computed score. A top-k query over defined sub-systems returns the objects with the k-highest aggregated scores under a monotonic function. Rank-aware query processing has become a vital need for many applications, especially in modern large-scale distributed applications as top-k queries enable fast, reliable and ad-hoc filtering of results. In this work, we study the problem of answering top-k queries over p2p networks under the general assumption that each peer maintains its own ranked data. The naïve solution to process a top-k query is to ask all peers to send their scores to the query originator, which merges all the results and returns the top-k ones. However, an efficient top-k algorithm should return the highest k results by limiting the messages transferred across peers, without examining all scores in a fixed number of rounds.

Taking into account the p2p context and these characteristics we seek efficient ways to answer global top-k queries that are issued in the underlying network. We focus mainly on the promising threshold-based techniques in order to retrieve the highest ranked results. This family of algorithms promises standard phases of termination which means limited transferred bytes and fast query answering, exact matching of results, easy setup and use as soon as peers have their data ranked according to the specified criteria. However, to the best of our knowledge, these algorithms have not been evaluated yet in p2p networks and it is not obvious whether they can be effectively used. From this family of algorithms, we take as a base a most promising algorithm called Hybrid Threshold (HT) [1] and adapt it to a p2p network structured according to a super-peer topology. The contributions of this paper are threefold. First, it presents the evaluation of exact threshold algorithms in large scale p2p networks with different data distributions across nodes. Second, it presents two extended versions of the HT algorithm which could be applied for peers that host various kinds of data in any p2p network supporting a super-peer topology for top-k query answering. Finally, the results of this study provide useful conclusions regarding the applicability of threshold techniques for current or future p2p data management applications. More details about this work can be found in [2]

2. Background and Related Work
The most prominent work in top-k query processing has been defined by the seminal approach of Fagin et al. on the Threshold Algorithm (TA) [3] which was designed for standalone database applications. But especially for p2p applications, TA seems insufficient since it requires several round-trips to return the results (unpredictable time and network consumption) because it doesn’t take data distributions into account. To the direction of TA adjustment for distributed networks, four algorithms TPUT [4] and {TPAT, TPOR, HT} [1] have been presented including 3 basic phases and using thresholds to order to determine the final top-k set of objects. These algorithms seem to overcome the problems of TA and belong to the class of exact algorithms which means that they return exactly the real highest ranked objects without any kind of prediction. TPUT prunes ineligible objects based on their scores in 3 standard phases and in its evaluation outperforms TA in most cases [1]. TPAT does not belong to the
category of exact algorithms since it uses statistics to further enhance the pruning power of TPUT. However, both TPAT and TPUT algorithm assume uniform data distribution which could be restrictive as, in real p2p systems some peers may host data following other distributions. TPOR prunes ineligible objects by their rankings (positions in sorted lists). The Hybrid Threshold Algorithm [1] combines the advantages of both TPUT and TPOR, estimates data distributions without a-priori knowledge, which means that it does not assume a specific distribution of scores. It is based on partial sums and upper bounds to prune non-eligible objects, and terminates in a fixed number of $3 + 1$ phases.

It is worth noting that all the above algorithms except TPAT belong to the family of exact algorithms, which means that they always return the highest ranked objects correctly without using any sense of probabilities. The latter refers to another category of algorithms usually termed approximate which enable the final pruning of objects under specific probabilistic guarantees is achieved using routing filters and histograms. A detailed presentation of exact, approximate and combined approaches can be found in [2]. Also in [2, 5] some related problems such as the query routing and the ranking of objects are addressed.

3. Methodology

3.1 Problem Statement & Main Adaptations

Following the query model of [1, 4] we assume that each peer maintains a list of pairs $(O, Si(O))$ where $O$ is an object and $Si(O)$ is the score of the corresponding object. From this point on, when we refer to object $O$ we mean the Object_id of $O$ and not the actual object. The objects (object_ids) in each list are sorted in descending order of their scores. If an object does not appear in the list of a peer, its score in that list is zero by default. After submitting the query to relevant peers according to the specified routing strategy, our goal is to find the $k$-highest aggregated values $(f(Si(O),...SM(O)))$, where $f$ is a monotonic function, which is used to compute the overall score of an object. We use the SUM function for ease of exposition. The objects with the $k$-highest values are denoted as $top-k$ objects.

All threshold algorithms are trying to use appropriate thresholds in order to prune ineligible objects with low aggregated scores and return the top-$k$ set of objects. In this paper, we study the problem of answering top-$k$ queries efficiently. An efficient top-$k$ retrieval algorithm in such context should primarily take into account the bandwidth consumption and the execution time as a real p2p system may receive thousands of queries at a time. Also some other characteristics need to be addressed, such as the ad-hoc behavior of peers, the different distributions of scores, the scalability and the topology of the network. A representative case of threshold algorithm is the HT [1] which resulted from a combination of TPAT and TPOR [1]. The first experiments on bandwidth consumption performance show that HT outperforms TPAT, TPOR and the alternative approach of TPUT [4] as well.

In this work we present two extended versions of the HT algorithm called HT-p2p and HT-p2p plus adapted to p2p environment. Due to lack of space we emphasize briefly on the main adaptations of these versions [2]. We choose a super-peer topology in order to avoid the flooding of network messages while the super-peers could play the role of central managers for each cluster of peers which they are responsible. Also, in the extended versions of HT choose to save the intermediate results and apply a specific policy for ad-hoc peers in order to be consistent. The storing ability of HT-p2p helps pruning some steps of the HT basic algorithm. The required data for saving are limited to seen object_ids, scores, partial sums which means that their capacity in bytes is fairly low. On the other hand we gain in processing cost (no need for examining again the “examined” seen objects) and communication cost as well (the pruning of some steps entails fewer transferred messages and less bandwidth consumption). Finally, for modeling reasons it is preferable to define discrete roles at peers for the HT-p2p algorithm. The peer that issues the original query across the p2p network is called an Originator peer. Respectively, its responsible super-peer plays the role of Originator super-peer. Each relevant to the query peer is called Contributor peer. A Collector super-peer executes the specific running instance of HT-p2p. It collects the intermediate results from the originator and contributors and returns them to the Originator super-peer. For the selection of the Collector super-peer we could take into account the number of Contributor peers or the number of the relevant objects.

3.2 The HT-p2p Algorithm

Assume that the Originator peer issues a top-$k$ query in the p2p network. The query is routed from Contributor super-peers to Contributor peers according to the applied routing strategy and these peers and their ranked objects are taken as input in the HT-p2p algorithm. An instance of HT-p2p is executed by the Collector super-peer to process the incoming top-$k$ query. The processing steps of HT-p2p are the following:

Phase 1: Each Contributor peer sends its top-$k$ objects to the Collector super-peer. The latter calculates the partial sums for all objects seen so far and identifies the objects with the $k$ highest partial sums. The Collector super-peer stores all intermediate results of this phase (seen objects, scores, and their partial sums). For an object $O$, the partial sum $SPsum(O) = S'1(O) + ... + S'm(O)$ where $S'(O) = S(O)$ if $O$ has been reported by peer $i$ to the super-peer, and $S'i(O) = 0$ otherwise. An object has been reported by a peer if it has been sent with its score to a super-peer at least once, so it has been stored.

Phase 2: The Collector super-peer sends the list $L$ and the threshold $T = t1/m$ to all the Contributor peers in the p2p network, where: $|L| = list$ of the top-k object IDs from the partial sum list, $\tau$ called “phase1 bottom” which is the $k$-th highest partial sum and $m$ is the number of peers at the specified cluster of super-peer1. Upon receiving the list $L$, for each object $O_j$ in $L$: peer $i$ finds its local score termed $Sij$, and determines the lowest local score $S_{lowest}$ among all the $k$ objects in $L$. If the object $O_j$ does not occur in the list of peer $i$ then $S_{ij} = 0$. Then peer $i$ sends the list of local objects whose values are greater than or equal to $Ti$, where $T = max(S_{lowest}(i), T)$, to the collector super-peer. If a score for an object of a specified peer has been sent in previous phase there is no need for the peer to send it again. Now the super-peer calculates the partial sums for all the objects seen so far, and identifies the objects with the $k$ highest partial sums. The kth highest partial sum called “phase2 bottom” and is denoted by $t2$. The Collector Super-peer denotes $Tpatch = t2/m$. If ($Ti > Tpatch$) the collector Super-peer sends these objects (with the k-highest partial sums) as top-k objects to the originator super-peer which returns them to the originator peer and the algorithm terminates. But if ($Ti < Tpatch$), then two additional phases (Phase 3 & 4) are needed for each peer $i$ where the above condition is true. The Collector super-peer stores all the intermediate results of this phase (seen objects, their scores, their partial sums).
Phase 3 (patch phase if necessary): The Collector super-peer sends $T_{\text{patch}}$ to peer $i$ as the threshold and asks for all objects whose scores are no less than $T_{\text{patch}}$ to be sent. Now the super-peer calculates the partial sums for all the objects seen so far, and identifies the objects with the k highest partial sums. The kth highest partial sum called “phase-3 bottom” and is denoted by $\tau_3$. Then the super-peer tries to prune away more objects by calculating the upper bounds of the objects seen and have stored so far. An upper bound for an object $O$ ($U_{\text{sum}}(O)$) is calculated by the formula: $U_{\text{sum}}(O) = S'^1(O) + S'^2(O) + \ldots + S'^m(O)$, where $S'_i(O) = S_i(O)$ if $O$ has been reported by node $i$; $S'_i(O) = \min(T_i, 1)$ otherwise. Then the super-peer removes any object $O_j$ from the candidate set whose upper bound is less than $\tau_3$ and returns the top-k candidate set.

Phase 4 (necessary if we run Phase 3): Since the Collector super-peer stores the intermediate results in this phase it just calculates the real scores for the top-k candidate set as it has been returned from the previous phase and then identifies the exact top-k objects. Finally it sends the top-k objects to the originator super-peer which returns them to the originator peer.

Accordingly to this algorithm, our ad-hoc policy in HT-p2p is the following: “When phase 1 of HT-p2p starts all the online contributor peers and their scores are considered. When a peer leaves the p2p network later, the algorithm continues to consider its data as they are valid despite the peer’s offline status. However, in this case for consistency reasons we reduce the number of available contributor peers which is used for determining $T_{\text{patch}}$ threshold. Also, since the top-k query is routed to a set of relevant contributor peers, if a new peer enters the p2p network during the execution of HT-p2p, it cannot be examined until the next relevant top-k query is issued.

3.3 The HT-p2p plus Algorithm

The HT-p2p plus is a distributed version of the HT-p2p. Having in mind that each super-peer may have under its cluster many thousands of peers, it is desirable for performance and scalability reasons to host one running instance of HT-p2p at each relevant super-peer which is executed independently of each other and in a distributed way. So instead of assigning all the set of peers to a single super-peer we use multiple super-peers and each one applies the HT-p2p only to the peers for which he is responsible for. Then the results from all super peers are combined to determine the final top-k objects. It is not necessary for all super-peers to perform the same number of steps during the execution of HTp2p but the final phase cannot begin until all the Super-Peers have calculated their final top-k items. In HT-p2p plus, an additional role is defined, namely that of Contributor super-peer, which contributes to the final top-k results by applying a running instance of HT-p2p across its relevant peers. In general, once a routing strategy has returned the set of ranked objects of relevant (contributor) peers for each corresponding (contributor) super-peer, an instance of HT-p2p plus is ready to run. If the Originator super-peer has the role of Contributor super-peer as well, then we select to give it the additional role of Collector. The role of Collector super-peer in HT-p2p plus is to collect all top-k results from the running instances of HT-p2p plus, combine them and finally return them to the Originator super-peer. The processing steps of HT-p2p plus are the same with HT-p2p for each independent running instance. So each group of super-peer and peers will execute the HT-p2p algorithm until the point that the top-k results have been calculated in the Super-Peer. After that instead of returning the results to the peer the initiated the query, each Contributor super-peer start communicate with the Collector super-peer for the last phase. This last phase is required for the combination of the results from the contributors’ super peers. At the end of Phase 4, the collector super-peer has to combine the results from all the contributor super peers. This has to do with the following processing steps of phase 5:

Each contributor super-peer sends its top-k objects to the Collector super-peer. The Collector super-peer combines all these objects and creates a list with discrete objects (L1). The collector super-peer sends this list to all the Contributor super-peers and asks the scores for these objects. When, all super-peers answer he calculates and stores the scores for all objects in L1 and identifies the objects with the k highest sums. The kth highest sum will be called “phase 5 bottom” ($\tau_5$). The Collector super-peer sets a threshold for phase-5 $T_{\text{combine}}=\tau_5 z$ (where $z$ is the number of the contributor super-peers) and announces (with a broadcast) this threshold to all the contributor super-peers. Then each contributor super-peer sends to the Collector super-peer the objects (and their scores) with scores greater than $T_{\text{combine}}$ so a new list with discrete objects can be created (L2) and stored. The final top-k objects will be calculated from the set $L_3=L_1 \cup L_2$. The collector super-peer has stored all the information for the objects in this list and in the previous steps and there is no need for further communication actions. After this point the Collector super-peer has identified the final top-k objects and can return them to the peer that initiated the query. The correctness of HT-p2p and HT-p2p plus is based on the native proof of [1], while the correctness of phase 5 can be found in [2].

4. Evaluation

For the evaluation the threshold algorithms behavior we measured the bandwidth usage and the total execution time. Each of the above was observed with the following criteria under consideration: the number of peers in the network, the number of objects located on each peer and the distribution that the score of the objects follow among the peers. For our experiments we generated 3 types of datasets that follow normal, uniform and zipfian distributions for a large number of peers, using the libraries of [6]. Also we collected manually some real datasets from imdb.com [7] as it contains rankings from real users to real objects (movies). We spread the scores of highly ranked movies across peers and we make the $IMDB$ spread dataset with unique and different (Object, Score) pairs following horizontal data partitioning. In all other datasets, as well, we produced synthetic data to evaluate cases where data were too or less similar among peers by using the random walk model. Thus each score at each peer $i$ is $s[i] = S[i-1] + C$, where $C$ is a constant. For all experiments we used the Planetisim Simulator [8]. For all simulated networks CHORD overlay [9] was used. The simulation was performed on a Pentium® 4 3.0GHZ 3.0GHZ with 1.5GB of RAM, Java version 1.6.0_10 under WINDOWS XP SP3.

Each time presented in the experiments, is the average time from 5 runs in safe mode. Also, in all experiments the top-10 items were asked from each algorithm.

Due to space limitations we present a part of the detailed evaluation on threshold algorithms [2]. We marked as HT-p2p plus BEST, the case in which the HT-p2p plus algorithm exhibits the best behavior while HT-p2p WORST is the corresponding worst case against the tested cases. As evaluating bandwidth performance all kind of datasets lead us to the same conclusions.

Figure 1 below show the results zipfian dataset $\{A=1, range of
scores from one to 500), while Figure 2 depicts HT-p2p plus cases for uniform dataset \( n = 150 \) values, range \( (1, 500) \) according to different network topology. For example, 4x5 in topology means that we had 4 contributor Super-Peers and 5 contributor peers. In both experiments we had a maximum of 75000 objects \((150 \text{ objects per contributor peer})\) across 500 peers.

From this kind of experiments, we can draw four main conclusions. Firstly, HT-p2p plus exhibits in most cases the best bandwidth performance. The differences between the HT-p2p-plus BEST and WORST were negligible. Secondly, the TPUT algorithm in all cases transferred more bytes than the naive algorithm. Specifically in all datasets for HT-p2p and HT-p2p plus algorithm we got about 100-497 % and 160-545 % less bandwidth consumption as compared with Naive and TPUT threshold algorithm respectively. For zipfian and uniform distributions the results were similar, but we measured sharper increases for HT-p2p plus algorithm, while in some cases of real “synthetic” data (IMDB), HT-p2p was better in bandwidth consumption. It is worth noting that when the HT-p2p plus BEST was second after HT-p2p, both algorithms were close enough. So, we can conclude that HT-p2p plus is in general the most economic for bandwidth usage. Finally for HT-p2p plus we noticed that for zipfian and uniform data distribution the more super-peers we used the less bytes transferred. Also, the algorithm finished in two running phases with identical results for best cases.

As evaluating time performance of algorithms we measured the total time each algorithm takes to return the final top-k results and the clear processing time which is the calculation time. Here we should mark that PlanetSim simulator cannot support a fully distributed message mechanism at Super-Peers and it is based on a random network topology. This means that it is not fair enough to compare HT-plus with the others who only need one Super-Peer. On the contrary, it is noteworthy that the HT-plus BEST is better in execution time than TPUT and not far from the HT-p2p which is faster. The low calculation times \((5.7 \% \text{ to } 18.9 \%)\) as compared with the total time) at each Super-Peer and this limitation of the simulator give us the opportunity to claim that a new fully distributed version of the simulator could bring HT-p2p plus (BEST) on the first position. In addition for both HT-p2p and plus, up to 200 peers, we have a linear increase in total execution time which becomes sharper if we increase the number of contributor peers to 400 or 500 (see figure 3). Finally TPUT appears to need more than 300% more time than HT-p2p and HT-p2p plus. On the other hand the naive case proved to be optimal for all datasets, two or three times faster than HT-p2p. The trade-off between time and network consumption arises clearly here.

5. Conclusion and Discussion

Exact Threshold-based top-k algorithms could be the solution for searching in modern p2p applications that maintain and use ranked data. The new extended versions HT-p2p and HT-p2p plus presented in this paper show promising results in terms of bandwidth consumption which is the most crucial parameter as the size of p2p networks increases. Ongoing work includes the precise assessment of the time performance of HT-plus in a fully-distributed simulation platform. It is also interesting to study the support of non-monotonic aggregation functions in order to enrich the query capabilities.

6. References

6. The SSFNET. http://www.ssfnet.org/javadoc/