TECHNICAL REPORT: ICS-FORTH/TR-407, JULY 2010, COPYRIGHT BY INSTITUTE OF COMPUTER SCIENCE, FOUNDATION FOR RESEARCH & TECHNOLOGY - HELLAS [2010]

A Detailed Evaluation of Threshold Algorithms for Answering Top-k queries in Peer-to-Peer Networks

Ioannis Chrysakis [1] Constantinos Chalkidis [1], Dimitris Plexousakis [1]

[1]Institute of Computer Science, FORTH, N. Plastira 100, Vassilika Vouton, GR-70013 Heraklion, Greece

{hrysakis@ics.forth.gr, conhalk@csd.uoc.gr, dp@ics.forth.gr}

Abstract. Ranking queries, also known as top-k queries, have drawn considerable attention due to their usability in various applications. Several algorithms have been proposed for the evaluation of top-k queries. A large percentage of them follow the Threshold Approach. In p2p networks, top-k query processing can provide a lot of advantages both in time and bandwidth consumption. 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 performance of Threshold Algorithms. 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.

Keywords: Top-k queries, query processing, peer-to-peer networks, distributed search and systems.

1 Introduction

With the amount of available data growing rapidly in various contexts, applications and scenarios, the need for answering queries effectively is increasing dramatically as well. The use of ranking queries provides a solution to the problem of effective search. It was first used for relational databases [1], [2], or standalone applications [3], but it can be also used for distributed applications like peer data management systems, which manipulate a huge amount of data while sharing information across peers [4]. Ranking queries, also known as top-k queries, produce results that are ordered on some computed score. A top-k query over defined subsystems 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. These data could be manipulated explicitly by peers (e.g. as in recommendation systems) where each user ranks its objects, or could be gathered by peers who receive information according to the context (e.g. in network monitoring, sensor networks, community mining). 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) [5] and adapt it to a p2p network structured according to a super-peer topology.

The contributions of this report are *threefold*. Firstly, it presents the evaluation of exact threshold algorithms in large scale p2p networks with different data distributions across nodes. In fact, both original HT [5] and TPUT [6] algorithms have been tested only as standalone applications (in one host computer, with native java simulation) for maximum numbers of 100 and 128 nodes respectively and without regarding execution times. Also TPUT assumes that all the scores are following only the uniform distribution which is not realistic. Additionally, there is no study directly comparing HT and TPUT neither in centralized nor in distributed settings. Secondly, this report presents two extended versions of the HT algorithm which could be applied for peers that host various kinds of data in any p2p network which could support 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.

The rest of the report is organized as follows. In section 2 we present the state of the art threshold-based algorithms suitable for answering top-k queries in distributed environments. We analyse their main advantages and compare them by extracting their qualitative characteristics. Also, we briefly mention the related algorithms which don't belong to this category of algorithms, some related research fields and problems and finally some recent approaches in the area of top-k query processing. In section 3 we present the general problem statement and our suggestions for the adaptation of previous threshold algorithms to p2p context. Then we present two extended versions of the Hybrid Threshold algorithm called HT-p2p and HT-p2p plus. The former is presented step by step with a simple example, while for the latter a proof of correctness is presented. In section 4 a detailed evaluation of top-k threshold techniques is presented including our new extended versions and in various scenarios in order to ascertain their efficiency. Section 5 presents our conclusions and some thoughts for discussion.

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 because it doesn't take data distributions into account. These characteristics cause TA to have unpredictable time and network consumption with a lot of messages transferred for each round-trip across the network, which constitutes a poor query answering performance. To the direction of TA adjustment for distributed networks, four algorithms TPUT [6] and {TPAT, TPOR, HT} [5] 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, like unbounded message rounds 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 [5]. 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 [5] 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. Performance comparisons regarding bandwidth consumption [5] showed that HT outperforms the other algorithms of this family. In table 1 we summarize the characteristics of these algorithms that belong to threshold-based top-k query processing techniques.

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 as **approximate algorithms.** These algorithms are based on the idea that, given a top-k query, a probabilistic guarantee that *"x percent of the retrieved objects are among the top-k objects we would get if we had asked all peers in the system"* can be provided. The final pruning of objects under specific probabilistic guarantees is achieved using routing filters and histograms. The main limitation of this approach is that it resorts to broadcasting when the desired number of results is too high or, when the user asks for a high degree of accuracy (approaches exactly the results like exact algorithms). Moreover, all algorithms of this family are based on TA [3], so they need several round-trips for retrieving the final results. The most promising approach of this family seems to be the KLEE framework [7] which terminates in a fixed number of communication rounds.

A different distributed approach is presented by Nejdl. et. al. [8, 9]. It combines ideas of semantic query routing based on indices and proposes a decentralized top-k query evaluation algorithm which is based on dynamically collected statistics put into local indices. However, the first time all peers have to participate in processing the

query. Then, several roundtrips are required for obtaining the final result. Also, in the case where the query is not contained in indices the algorithm needs more time and network bandwidth. In [10], some novel methods are presented for optimizing top-k aggregation queries for both exact and approximate algorithms based on choosing different thresholds for each peer (instead of a global threshold at each phase). It would be interesting for this work to evaluate more network factors such as bandwidth consumption and the ad-hoc behavior of algorithms when peers join or leave the p2p network frequently. The same authors extended their work in a first attempt to compare TPUT and KLEE in either exact or approximate mode with the suitable adjustments [11]. The results showed that the adjusted version TPUT is slightly preferable than that of KLEE considering bandwidth and response time performance.

Table 1. Comparison of top-k threshold query processing algorithms

Algorithm	**TPUT**	**TPAT**	**TPOR**	**HT**
Exact Matching of results	Yes	No	Yes	Yes
Assumes specific data distribution	Yes (uniform)	No	Yes (uniform)	No (works without having a-priori knowledge)
Communication Phases	3	3	3	3+1
Evaluated in p2p networks	No	No	No	No

3 Methodology

3.1 Motivation

Nowadays, the p2p model is used for diverse applications and services, including content storage, sharing (*file-sharing, content distribution, backup storage*) and communication *(voice, instant messages, multicast*) to name a few. The size of data is increasing rapidly at peers day by day, so the problem of effective search in peer-to-peer networks becomes more crucial than ever. This problem could be divided into two problems: query routing and query processing. First, we have to decide where to route each submitted query instead of broadcasting the query to the entire network which is the naïve case of routing. Secondly, we have to decide which peers should participate to the query processing stage. In this work we do not study the problem of how to route the query to relevant peers, but how we can process efficiently the query using top-k query processing techniques upon our ranked data.

3.2 Data Model and Problem Statement

Following the query model of [5, 6] 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 algorithm manipulates (sends, receives etc.) object identifiers. 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(S1(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*. Objects can be thought of as, e.g., RDF resources if peers host RDF/S data or tuples if they host relational databases. Each object is scored according to the selected scoring technique which in turn determines the applied function accordingly to the specified needs of each p2p application. For example we could use a weighted monotonic function for the computation of the final score for each object like the formula: *Final Score = w1*s1 + w2*s2 + w3*s3 where* w1, w2, w3 could be some predefined weights for each property according to the preferences of the user who sends the query to the system. For our algorithms all scores for discrete objects are taken as input to calculate the top-k set.

All threshold algorithms are trying to use appropriate thresholds in order to prune some ineligible objects with low aggregated scores and return finally the top-k set of objects. In this report, we study the problem of answering top-k queries efficiently. An efficient top-k retrieval algorithm in such context should take into account first of all the bandwidth consumption and the execution time as a real p2p system may receive thousands of queries per 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 representing case of threshold algorithm is the HT [5] which resulted from a combination of TPAT and TPOR [5]. The first experiments on bandwidth consumption performance showed that HT outperforms TPAT, TPOR and the alternative approach of TPUT [6] as well. Thus in the next subsection, we try to verify whether the HT algorithm could be fully adapted from middleware to p2p environment emphasizing the main directions for all threshold algorithms. To the best of our knowledge, none of the algorithms of this family has been adapted and tested in p2p environment.

3.3 Main Directions and Adaptations

Due to the aggregation nature of top-k problem, it is obvious that a central manager is required (to gather the results and forward the answers), as well as, a suitable network topology in order to avoid flooding of network messages. *A super-peer topology* combines these issues as super-peers play the role of central managers for each cluster of peers for which they are responsible. One idea for this classification of peers could be by semantic criteria, similarly to the idea of Semantic Overlay Networks [17]. Thus by applying a suitable routing mechanism which determines the

relevant peers each super-peer should process only a subset of queries which are semantically matched. The problem of query routing as we mentioned before is out of the scope, although with techniques like [13] a query routing algorithm could be combined efficiently. Thus, the extended versions of HT are built upon this super-peer topology and could be supported generally by all the threshold algorithms.

Having in mind that peers could join or leave the network frequently, the top-k algorithm should first take into account this characteristic in order to define a consistent policy. Secondly it could store some intermediate results in order to enhance the whole processing technique. The HT algorithm processes the incoming queries on the fly and does not study the case when a peer enters or leaves the network during its execution. In the extended version HT-p2p we 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 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 all the contributor peers 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. In the next subsections we present the extended versions of HT called HT-p2p and HT-p2p plus which use these terminology concepts. Before presenting analytically the extended versions of HT we present a brief comparison of all versions based on their main differences.

Table 2. Comparison of all version of the HT algorithm

Algorithm	**HT (Original)**	**HT-p2p**	**HT-p2p plus**
Network Topology	Unstructured	Super-Peer	Super-Peer
Storing ability of intermediate results	No	Yes	Yes
Fully Distributed	No	No	Yes
Evaluated in p2p networks	No	Yes	Yes

3.4 The HT-p2p Algorithm Step By Step

Let's assume that the Collector super-peer is SP1 and the relevant peers according to the routing strategy are: Peer1, Peer2, and Peer3 (m=3). These peers are the contributors to a sample query. Let's assume that the Originator peer is peer1, so SP1 is also the Originator super-peer and we are looking for a top-2 query (k = 2). Table 1 below depicts the (Object_id, Score) pairs at each peer of SP1. The steps performed by HT-p2p for the sample top-2 query are the following:

Table 3. (Object_id, Score) pairs at each peer of SP1

Peer1	Peer2	Peer3
(O4, 21)	(O5, 32)	(O3, 30)
(O2, 17)	(O1, 29)	(O5, 14)
(O5, 11)	(O18, 29)	(O18, 9)
(O3, 11)	(O3, 26)	(O4, 7)
(O6, 10)	(O9, 20)	(O2, 1)
(O7, 10)	(O4, 9)	(O8, 1)
(O11, 8)	(O14, 5)	
(O12, 6)	(O16, 2)	
(O15, 6)	(O13, 1)	
(O13, 4)		

- In **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, their scores, and their partial sums). For an object O, the partial sum $Spsum(O) = S'_1(O) + ... + S'_m(O)$ where $S'_i(O) = Si(O)$ 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.
- *Thus, Peer1 sends its top-2 objects with its corresponding scores to SP1: (O4, 21), (O2, 17). Peer2 sends respectively (O5, 34), (O1, 29) to SP1 and Peer3 sends (O3, 30), (O5, 14). Then SP1 calculates the partial sums (Spsum) for all seen objects: Spsum(O4) = 21, Spsum(O2) = 17, Spsum(O5) = 48, Spsum(O1) = 29, Spsum(O3) = 30. The k=2 highest partial sums belong to O5, O3 and their value is 46, 30 respectively. SP1 stores all the intermediate results of this phase.*
- **In Phase 2** the Collector super-peer sends the list L and the threshold T = τ1/m to all the Contributor peers in the p2p network, where: {*L = list* of the top-k object IDs from the partial sum list, τ1 called *"phase1 bottom"* = the k-th highest partial sum and m = the number of peers at the specified cluster of super-peer}. As per Phase 1, τ1 = 30 and the list L contains O5, O3. Thus SP1 broadcasts the list L = {O5, O3} and the threshold T =10 since it is equal to the fraction: τ1 / m where m = 3.
- Upon receiving the list L, for each object O_j in L: peer i finds its local score termed S_{ij}, and determines the lowest local score *S_lowest (i)* among all the k objects in L.

If the object Oj 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 T_i, where *Ti=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.

- *Consequently, at peer1 the lowest local score is 11 and it is coming both from object O3 and O5. At peer2 the lowest local score is 26 (object O3), whereas at peer3 the lowest local score is 14 (object O5). For peer1, T1=11 so it sends objects (O5, O3) to SP1 with their scores, as long as (O4, O2) have been sent in phase1and as soon as these values are greater than Ti. For peer2 T2=26, so it sends objects (O18, O3) with their scores as long as (O5, O1) have been sent in previous phase. For peer3 there is no need for sending any pairs since T3=14 and to SP1 the pairs (O3, 30), (O5, 14) had been sent earlier in phase1.*

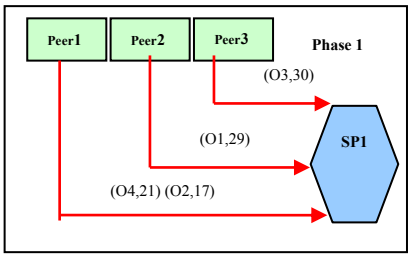

Fig. 1. Transferred data in Phase 1

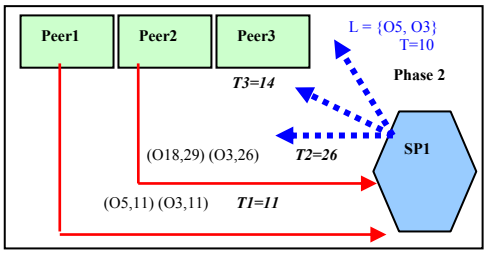

Fig. 2. Transferred data in Phase 2

- 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-2 bottom"* and is denoted by τ2. The Collector super-peer denotes *Tpatch = τ2 /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).
- *Thus, the partial sums for the seen objects are: Spsum(O4) = 21, Spsum(O2) = 17, Spsum(O5) = 57, Spsum(O3) = 67, Spsum(O1) = 29, Spsum(O18) = 29. Thus, the 2 highest are Spsum(O3) and Spsum(O4) where the last is equal with τ2 since it is the kth. Therefore Tpatch = τ2 / m = 57 / 3 = 19.For peer1 and peer3 there is no such need for patch phase because their thresholds (T1=11, T3=14) are less than Tpatch. For peer 2 we need to execute a patch phase since T2 =26 >19.*
- In **Phase 3** which is not always necessary, the Collector super-peer sends Tpatch to peer i as the threshold and asks for all objects whose scores are no less than Tpatch 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 τ3 threshold.

- *Thus, SP1 requests from peer2 to send all its objects that are no less than Tpatch = 19. Thus peer2 sends {O4, O14, O16, O13} and the respective scores. The partial sums for the current seen objects are: $S_{psum}(O18) = 29$, $S_{psum}(O1) = 29$, $S_{psum}(O2) = 17$, $S_{psum}(O3) = 67$, $S_{psum}(O5) = 57$, $S_{psum}(O4) = 21$, $S_{psum}(O13) = 1$, $S_{psum}(O14) = 5$, $S_{psum}(O16) = 2$.*
- 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 (Usum(O)) is calculated by the formula: $Usum(O) = S'_1(O) + S'_2(O), + \ldots S'_m(O)$, where: $S'_i(O) = S_i(O)$ if O has been reported by peer i; $S'_i(O) = \min(T_i, Tpatch)$ 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.
- *In the example, SP1 calculate the upper bounds for all seen objects: S'(O18) = 54, S'(O1) = 54, S'(O2) = 50, S'(O3) = 67, S'(O5) = 57, S'(O4) = 44, S'(13) = 26, S'(14) = 30, S'(16) = 27. So, SP1 prunes O18, O1, O2, O4, O13, O14, O16 objects from the top-k candidate set, since their upper bounds are less than τ3 = 57.*
- The **Phase 4** is necessary in the case when we have 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.
- *In the example, since the top-k candidate set from phase 3 contains exactly k=2 objects there is no need to calculate the real scores for these objects to determine the highest ones, so SP1 which is the Originator super-peer returns O3 and O5 to peer1 (Originator peer) as top-k objects.*

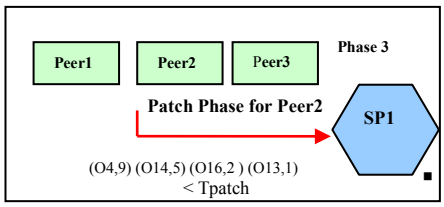

Fig. 3. Transferred data in Phase 3

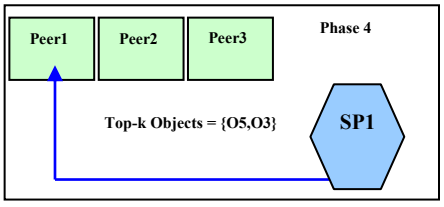

Fig. 4. Transferred data in Phase 4

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 Tpatch threshold. The algorithm continues normally in the next steps. 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. If we would like to take into account the new peer, the specified routing strategy should examine whether this peer is relevant to the top-k query and in that case it should send the query to it. But this means that in practice the top-k algorithm should restart since all the relevant parameters would need to be recomputed (thresholds, number of relevant contributor

peers etc). Obviously this alternative policy requires more time, network bandwidth and maybe causes situations of frequent restarting which surely constitutes an inefficient policy.

3.5 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 HTp2p 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 calculate 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. Otherwise, a Collector super-peer can be any of the *Contributor super-peers*. 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 return them to the Originator super-peer.

The processing steps of *HT-p2p plus* are the same with HTp2p 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 will 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. In this phase only the super peers are required to exchange messages. At the end of Phase 4, the collector super-peer has to combine the results from all the contributor super peers. The *processing steps of **phase 5** are the following.*

- 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"* (*τ5)*.

- The collector super-peer sets a threshold for phase-5 $T_{combine} = {}^{\tau 5}/_{z}$ (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 $T_{combine}$ so a new list with discrete objects can be created (L2) and stored.
- The final top-k objects will be calculated from the set L3= {L1 ∩ L2}. 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. It is important that during the merging of the results from the Contributor super-peers we will not conclude to incorrect results. The correctness of HT-p2p and HT-p2p plus is based on the native proof of [5]. However, the following paragraph explains why the steps of phase 5 should always return a correct candidate set of objects.

Lemma 1

Phase 5 will **include** in the final candidate set any Object with score high enough to be a top-k object even if it is not present in the initial L1 list.

Proof:

Assume that we have z in number super-peers and A_b is the ***phase 5 bottom***. Also we assume that A_1, A_2 ,A_3,.....,A_z are the kth scores from each super peer (each score can belong to a different Object). It is possible that there will be an object with scores $O_1 < A_1$, $O2 < A_2$, $O_3 < A_3$ ... but with total score=O_1+O_2+...+O_z greater than Ab. We will show that the algorithm will include this object in the final candidate top-k set of objects.

Assume that O has final scores $O_1 < A_1$, $O_2 < A_2$ …, $Oz < A_z$ in each Super Peer. Assume also, that the sum O_1+ O_2 +O_3+…+O_z > A_b (1) $A_b$can be expressed as a sum of z in number of $T_{combine}$ .So (1) is equivalent with$\sum_1^z ScoreFromSuperPeer_i(O) < T_{combine} * z$. So at least one the final scores $O_{1,}$..O_z has to be greater than $T_{combine}$ (if $T_{combine}$ > O_i for each Super Peer then A_b will be greater than the sum of final scores O_1+ O_2 +O_3+…+O_z and we assumed it is not (1)). So since we have at least one score O_i which is greater than $T_{combine}$ the algorithm will **include it** in the final set.

Lemma 2

Any object left out of the candidate set **cannot be** a TOP-K object

Proof:

Assume z in number super-peers for an object O which is not included to the final candidate set must be:

$ScoreFromSuperPeer_i(O) < T_{combine} \forall i, 1 \leq i \leq z$. ***(1)***
So $\sum_1^z ScoreFromSuperPeer_i(O) < T_{combine} * z$ *meaning that TotalScoreForObject(O) < phase 5 bottom* ***(2)***.
But, always: *phase 5 bottom* ≤ *The final k highest score* **(3)**. From (2) and (3) we conclude that O is not a top-k object.

4 Evaluation

4.1 Evaluation of algorithms in p2p networks.

The evaluation of data-centric algorithms like top-k threshold query processing algorithms could be done either by implementing all algorithms upon a network platform or by using a network simulator. One of the most famous and widely used network platforms is JXTA [18]. It provides a common set of open protocols and an open source reference implementation for developing general purpose, interoperable and large scale P2P applications. However by using JXTA the supported scalability of peers is limited to less than 50 peers with the use of evaluation techniques like [13]. In any case, it is obvious that we need a more powerful and flexible tool in order to run our algorithms simultaneously upon large-scale p2p networks, to draw safe conclusions. This could be done by simulating all these algorithms which is far more efficient and less costly than actual implementations and without the need strong computational power. A variety of simulators can be found in the literature or in the web for the evaluation of algorithms in p2p networks; each of them has been designed for specific purposes [19]. In fact the majority of network simulators belong to network-oriented or designed for specific needs .Especially for top-k threshold query processing algorithms we conclude that the four simulators PlanetSim [20] ,Overlayweaver [21], PeerSim [22] and P2psim [23] seem to adapt in our case, so we evaluate them based on the following criteria:

- **Architecture.** It is related to the design and the functionality of the simulator. In other words it is the way the features have been implemented.
- **Usability.** Refers to how convenient the simulator is to use and understand. This is closely related with the documentation, API and the activity of the corresponding community support.
- **Scalability.** Refers to how many nodes can be supported and how the simulator scales when the nodes are increasing.
- **Ability to host applications** in nodes. This means that we could run any algorithm in each node with a few adaptations, instead of rewriting all algorithms to simulator API and language.
- **Statistics**. The ability to gather the results and produce statistics for our experiments. More specifically, the measured bandwidth over the network, the numbers of messages send from each node and total execution times are the most desired features for monitoring algorithms.

Table 4. Evaluation of p2p simulators

Simulator	Architecture	Usability	Scalability #peers	Host applications	Statistics
P2PSim	Discrete-event for structured P2P networks	Poor documentation	3000	Yes (C++)	Limited set of statistics
Overlay Weaver	Distributed Emulation	Good API but very poor documentation	150000	Yes (Java)	N/A
PlanetSim	Discrete-event simulation, distinct separation of services and overlay	Good tutorials ,Very good API and documentation	100000	Yes (Java)	Supports Statistics & Visualizer
PeerSim	Query-Cycle or Discrete-event simulation	Only query-cycle is documented	10^6	N/A	Requires implementation of components

Our study leads us to the selection of PlanetSim [20] simulator although the majority of these algorithms could not support fully simultaneous distributed routing of messages. The PlanetSim's architecture (see figure 5 below) comprises three main extension layers constructed one on top of another. Applications are built in the upper layer using the standard Common API facade. This facade is built on the routing services offered by the underlying overlay layer. Besides, the overlay layer obtains proximity information to other nodes asking information to the Network layer. The Simulator dictates the overall life cycle of the framework by calling the appropriate methods in the overlay's Node and obtaining routing information to dispatch messages through the Network.

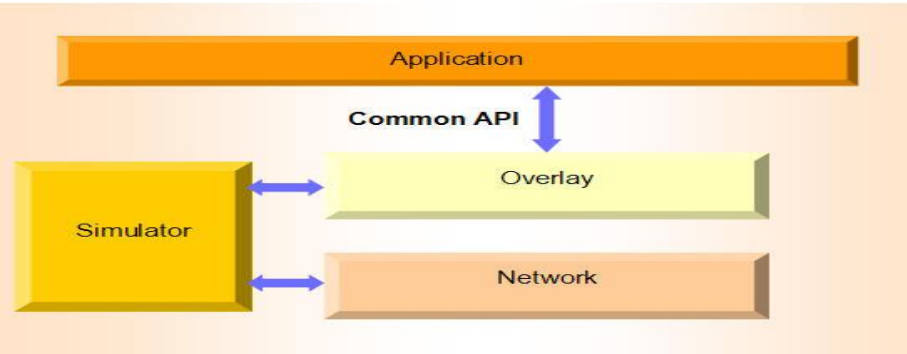

Fig. 5. The PlanetSim's Core Architecture

4.2 Evaluation Setup

The goal of this evaluation is to demonstrate each algorithm's behavior, under various circumstances which would be of important significance as top-k query

algorithms are highly important in modern p2p systems. The three main aspects that were used for the evaluation of each algorithm were the following: The *bandwidth usage, the total (execution) time and the number of objects accessed (seen objects)*. The network and time factors are used for the quantitative evaluation, while the objects factor is used for qualitative evaluation. 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 objects follow among the peers. We also measured the time in which the "*Super-Peer*" needs to store and process the results which receive from the peers in each phase. We named this unit *calculation time*.

Some previous works have assumed that the ranked data following a uniform distribution across the nodes. However this is not realistic enough, so more datasets are needed in order to carefully evaluate each algorithm. Also little research has been done around the scalability of the algorithms when the number of nodes is increasing. 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 [24]. Also we collected manually some real datasets from imdb.com [25] 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*. In this dataset all peers contain 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 Planetsim version 3.0 was used as the simulation tool. For all simulated networks *CHORD overlay* [26] was used. The simulation was performed on a *Pentium® 4 3.0GHZ 3.0GHZ with 1.5GB of RAM.* The Java version was *1.6.0_10* under *WINDOWS XP SP3*. For each experiment the computer was in safe mode. Each time presented in the experiments, is the average time from 5 runs. Also, in all experiments the *top-10 items* were asked from each algorithm.

4.2 Experiments

4.2.1. Evaluating bandwidth consumption.

The intention of the **first experiment** was to compare threshold algorithms in terms of *bandwidth consumption*. As we talk about an algorithm that process online data across the network it is crucial to study this parameter which affects the general efficiency of the whole top-k query processing technique. Surely since the top-k threshold algorithms transfer messages which include (Object_id, Score) pairs and some control information, this means that we don't expect too large bandwidth consumption. On the other hand our intention is to compare all the relative algorithms of this family to check from this aspect their scalability and efficiency.

Thus, we measured for a given set of peers, the size of transferred messages each algorithm uses until it returns the final results. We tested *HT-p2p*, *TPUT*, *Naive* algorithm and the *HT-p2p plus* for different cases of peers per Super-Peers. We marked as *HT-p2p plus BEST*, the case in which the HT-p2p plus algorithm exhibits

the best behaviour while *HT-p2p WORST* is the corresponding worst case against the tested cases. All datasets lead us to the similar conclusions about the network performance of threshold algorithms in top-k query processing.

We present the results of three datasets, the *zipfian dataset* {A=1, range of scores from one to 500}, uniform {n=150 values, range= (1, 500)} and the IMDB real synthetic dataset. In the first and the second case we had a maximum of *75000 objects* (150 objects per contributor peer) across 500 peers, where in the second we had *100000 objects* (250 objects per contributor peer) across 400 peers. Our first observation could be clearly that the values between BEST and WORST case were close enough. Figures 6, 8, 10 and table 5, 6, 7 below present these results. We also present the different running cases of HT-p2p plus according to the network topology. (see figures 7, 9, 11) For example, 4x5 in topology means that we had 4 contributor Super-Peers and 5 contributor peers running the algorithm.

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. In general HT-p2p plus and HT-p2p manage to reduce the bandwidth considerably. 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.

Our last but not least conclusion about HT-p2p plus and the different running cases is that for zipfian and uniform data distribution the more super-peers we used the less bytes transferred. In these cases the algorithm finished in two running phases. Is is noteworthy that all best cases for this kind of data were identical. But for IMDB data, where we noticed some patch phases (i.e two additional communication phases required for each running instance of Contributor Super-Peer) we could not claim the same argument as we got different values for best and worst cases for different datasets of HT-p2p plus. Consequently the network structure affects clearly the bandwidth consumption only if we run the standard phases of the algorithm. Maybe more experiments with different simulators or collected datasets are required to draw a safer conclusion about the cases when patch phases are necessary.

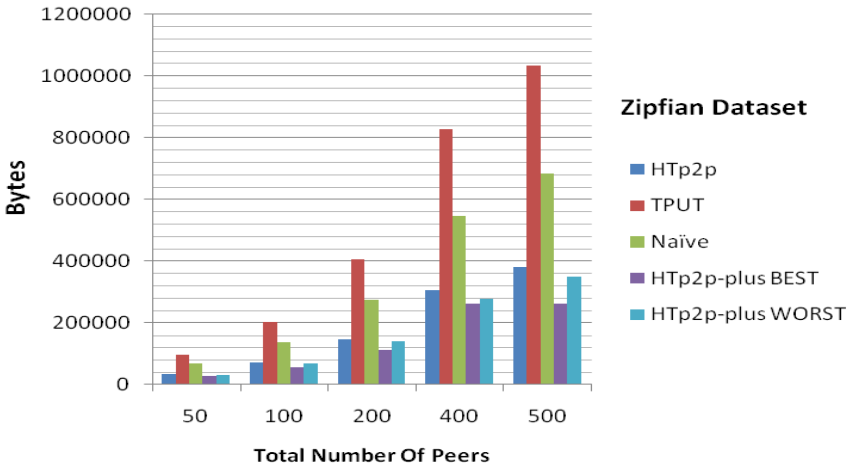

Fig. 6 Total number of transferred bytes for zipfian dataset

Table 5. Transferred Bytes for Zipfian Dataset

Peers	**HT-p2p**	**TPUT**	**Naive**	**HT-p2p BEST / Topology**	**HT-p2p WORST / Topology**
20	12246	37147	27329	**10644** /4x5	11157 /2x10
50	33855	98063	68282	**27898** /5x10	32304 /2x25
100	72123	201860	136554	**55662** /10x10	67397 /2x50
200	146178	406095	273122	**110820** /20x10	139926 /2x100
400	304448	826644	546352	**260519** /10x40	278563 /4x100
500	380283	1032800	682873	**264833** /100x5	349610 /5x100

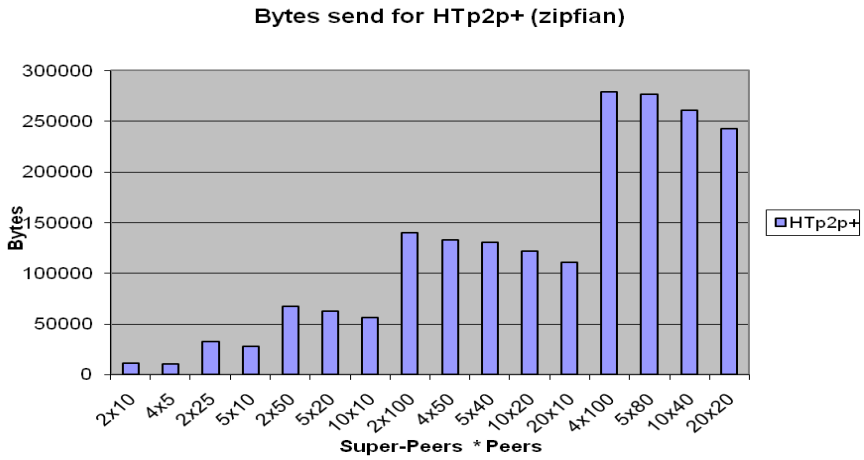

Fig. 7 HT-p2p plus cases for zipfian dataset

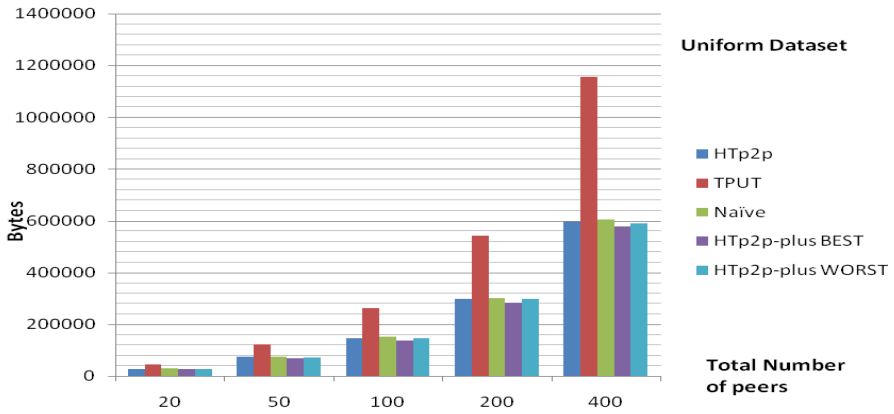

Fig. 8 Total number of transferred bytes

Table 6. Transferred Bytes for Uniform Dataset

Peers	HT-p2p	TPUT	Naive	HT-p2p BEST / Topology	HT-p2p WORST / Topology
20	**27511**	44024	30205	27563 /4x5	28013 /2x10
50	73477	121243	75525	**68871** /5x10	71137 /2x25
100	146915	261702	151026	**137579**/10x10	146091 /2x50
200	297204	541151	302032	**283504**/20x10	296797/2x100
400	595946	1155606	603904	**578282**/10x40	590383 /5x80
500	750497	1500936	754899	**674330**/100x5	740945/5x100

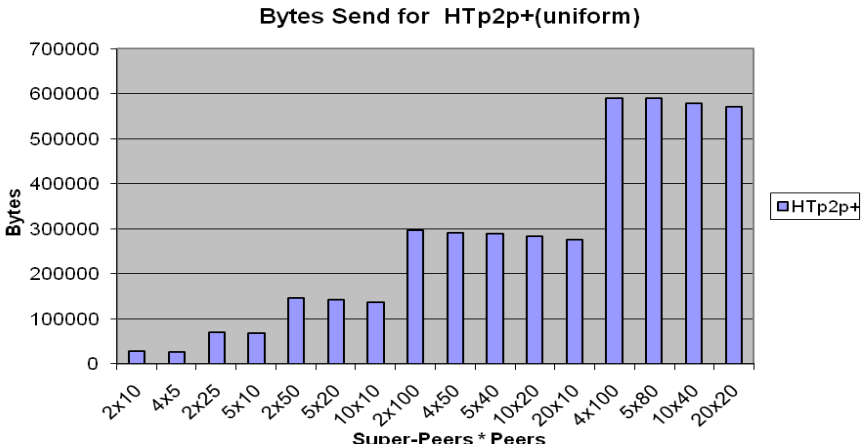

Fig. 9 HT-p2p plus cases for uniform dataset

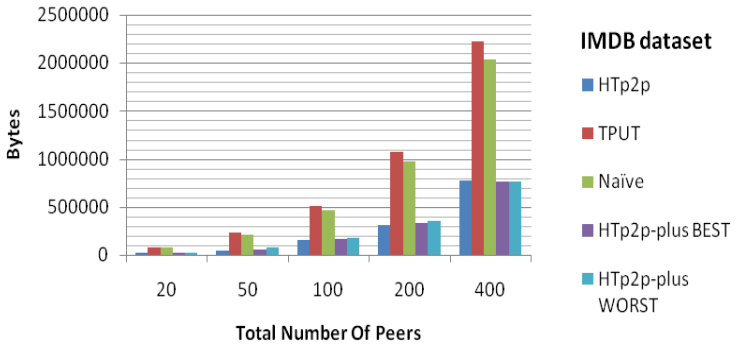

Fig. 10 HT-p2p for IMDB dataset

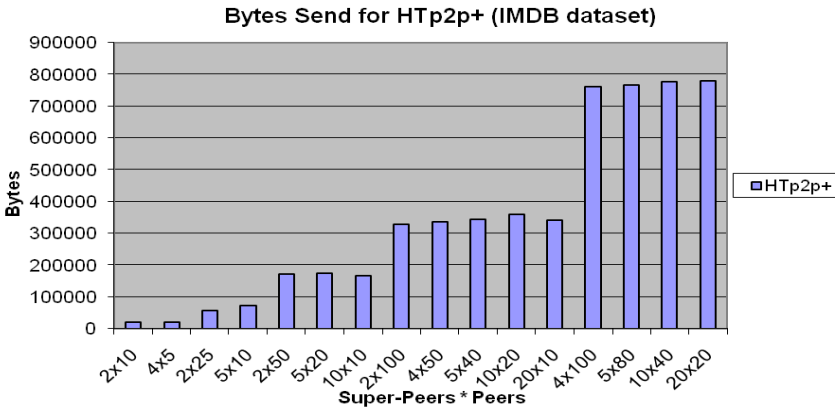

Fig. 11 HT-p2p plus cases for IMDB synthetic dataset

Table 7. Transferred Bytes for IMDB Synthetic Dataset

Peers	**HT-p2p**	**TPUT**	**Naive**	**HT-p2p BEST / Topology**	**HT-p2p WORST / Topology**
20	20077	80755	74165	**19994** / 4x5	20012 / 2x10
50	**43222**	235767	214902	56472 /2x25	72645 / 5x10
100	**151228**	505026	460039	166022 /10x10	175159 /5x20
200	**313251**	1070057	976104	328924 /2x100	358702 /10x20
400	771749	2223911	2039857	**762621** /4x100	767427 / 5x80

4.2.2. Evaluating execution time performance.

In the **second experiment** we measured the execution time each algorithm takes to return the final top-k results. In this experiment 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*. In fact the calculation time, is the time the Super-Peer(s) spend on sorting-calculating- "processing" the items-scores, that receives from the peers. Here we should make an **important note** about PlanetSim simulator. It 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 just 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 the optimal. The low calculation times 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. We present in this subsection representative results of the real IMDB datasets and the zipfian dataset in figure 12 and 13 respectively. The main difference besides these cases is that in the first case the HT-p2p algorithm needs mostly some patch phases to complete its execution, while in zipfian dataset in all cases it the results returned just in two phases of the algorithm. This could explain firstly the large differences at times for the same size of peers. Especially, for IMDB dataset the sorted order of time-efficiency is: *{Naive, HT-p2p, HT-p2p BEST, HT-p2p WORST, TPUT}* in all cases of peers. But for zipfian datasets the order for 20, 50,100,200,400 and 500 peers was not the same:

- {*Naive, HT-p2p BEST, HT-p2p, TPUT, HT-p2p WORST}*
- *{Naive, HT-p2p BEST, HT-p2p, HT-p2p WORST, TPUT}*
- *{Naive, HT-p2p, HT-p2p BEST, HT-p2p WORST, TPUT}*
- *{Naive, HT-p2p, HT-p2p BEST, TPUT, HT-p2p WORST}*
- *{Naive, HT-p2p, HT-p2p BEST, TPUT, HT-p2p WORST}*

As we could see in the first two cases the distributed HT-p2p BEST finished before HT-p2p while in the last two phases the HT-p2p WORST presented the worst of behaviour on time performance. In these worst cases the number of Contributor Super-Peers was much greater than the contributor peers (50x8 and 100x5).

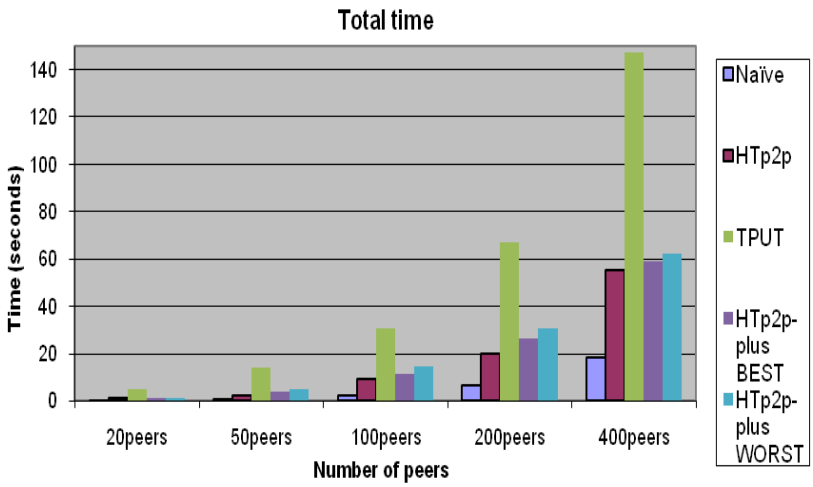

Fig. 12 Total time of execution in seconds for IMDB dataset

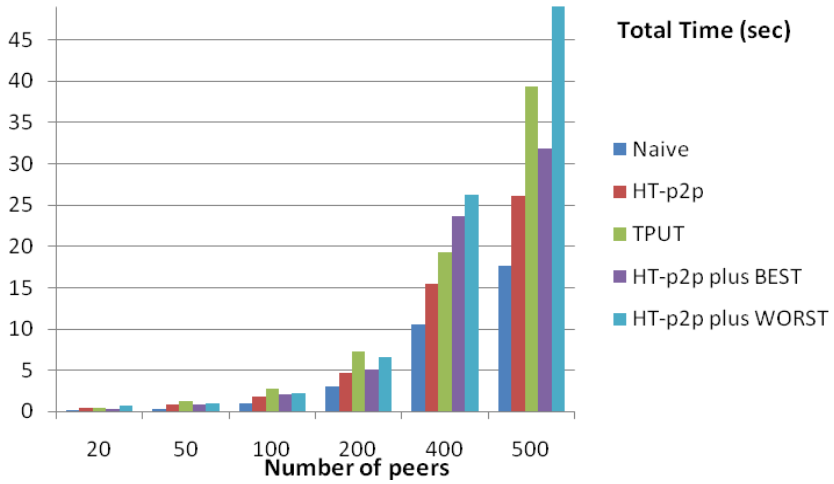

Fig. 13 Total execution time in sec for zipfian dataset (no any patch phase appeared)

As comparing total execution time against calculation time in HT-p2p plus we could say clearly that most of the time is spent in sending/receiving messages, which could

be caused by the simulator (see figure 14 and table 8 below). In this figure which corresponds to the case when we had the maximum number of Super-Peers for zipfian dataset, the percentage of calculation time through the total time is varying from 5.7 % to 18.9%, which is the worst case for calculation time. It is noteworthy that as we the size of participating peers increases the calculation time falls dramatically in most of the cases. Thus, the distributed nature of HT-p2p proved to share the calculation cost among peers due to load balancing at contributor Super-Peers. In table 9 suggestively, we can view the specific results of the representable HT-p2p plus best case 4x100 of calculation times at each Super-Peer using IMDB real synthetic data. It is clear that at this experiment the calculation time is taking just the *1%* of the total time. But this rate in other datasets (zipfian, normal, IMDB spread) can take up to 18.9 % of the total time.

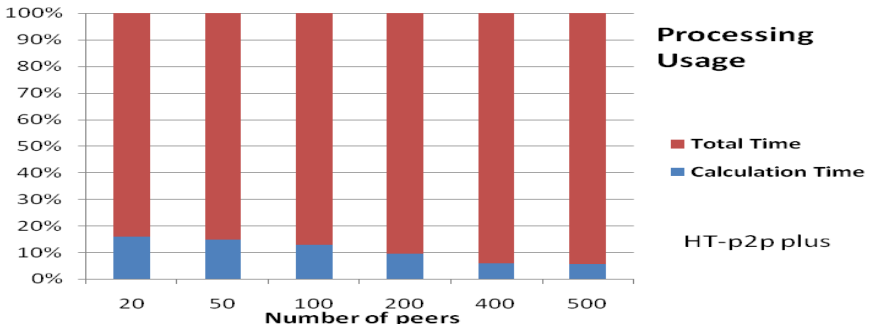

Fig. 14 Calculation vs. total execution time for zipfian dataset for HT-p2p plus

Table 8. Calculation time of execution in seconds for zipfian dataset in HT-p2p plus

Peers	**20**	**50**	**100**	**200**	**400**	**500**
HT-p2p plus Calculation Time	0,0801	0,178	0,330	0,688	1,620	2,825
HT-p2p plus Total Time	0,4224	1,031	2,268	6,559	26,209	49,180
Used Super-Peers	4	5	10	20	50	100
Percentage of Calculation Usage	18,966	17,3	14,55	10,48	6,18	5,74

Table 9. Calculation Times of 4x100 case for HT-p2p plus BEST using IMDB data

Super-Peer ID	Calculation Time (sec)
1	0,122
2	0,1652
3	0,1196
4	0,1108
Total Calculation/ Full Execution Time (s)	0,6262 / 62,196

Four *main conclusions* could arise from this experiment. Firstly, the naive case proved to be optimal for all datasets, two or three times faster than HT-p2p. Surely for this result we should take into account two things. The first has to do with the Planetsim limitation of distributed running and the other has to do that the naive algorithm presented the maximal bandwidth consumption which is not surely preferable. Secondly 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. Thirdly, due to the nature of the simulator HT-p2p needs the least time to return the final results, but, most of the times HT-p2p plus, due to load balancing, needs fewer time to process the scores. Finally TPUT appears to need more than *300%* more time than HT-p2p and HT-p2p plus. The *trade-off* between time and network consumption arises clearly from this experiment.

4.2.3. Accessed objects – Seen Objects.

We also checked in the **third experiment** how many objects were accessed from databases to retrieve their scores termed as the *seen objects*. We check substantially HT-p2p vs. TPUT with this qualitative characteristic as soon as the HT-p2p plus presents the same behaviour with the HT-p2p while naive case includes the maximum number of objects. We noticed that all basic threshold algorithms (HT-p2p, TPUT) access the more than *80%* of the objects for all 0datasets. In some cases HT-p2p and TPUT had the ideal behaviour (IMDB spread dataset, see figure 15) but in other datasets HT-p2p is better (IMDB synthetic, normal, zipfian), see fig.16. TPUT presented the better behaviour when we had the maximum number of scores per peer (1000 items) for normal or uniform data distribution. The latter corresponds to the best case for TPUT (see figure 17), something which was expected as soon as it is designed for this kind of scoring distributions. But, in most of the other cases TPUT has the maximum number of seen objects which is identical to the naive algorithm. When we used synthetic and similar enough datasets the seen objects were reduced dramatically for HT-p2p and HT-p2p plus. Thus for the normal synthetic case the HT-p2p algorithm examined as seen objects the actual top-10 ones. From the other hand in this case TPUT examined all the items which is identical to the naïve case. The *general conclusion* here is that for similar synthetic data HT-p2p has the best

behaviour. On the other hand, the high rates in seen objects is a topic for further study and discussion to discover maybe more advanced methods for defining thresholds maybe by using specified peer thresholds instead of global thresholds with techniques such as the ones of [10].

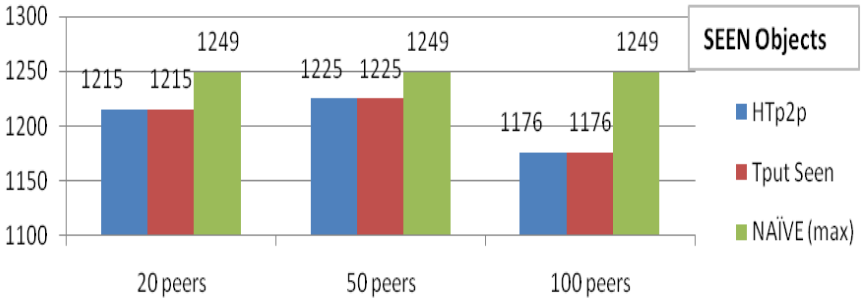

Fig. 15 Seen objects for IMDB spread dataset

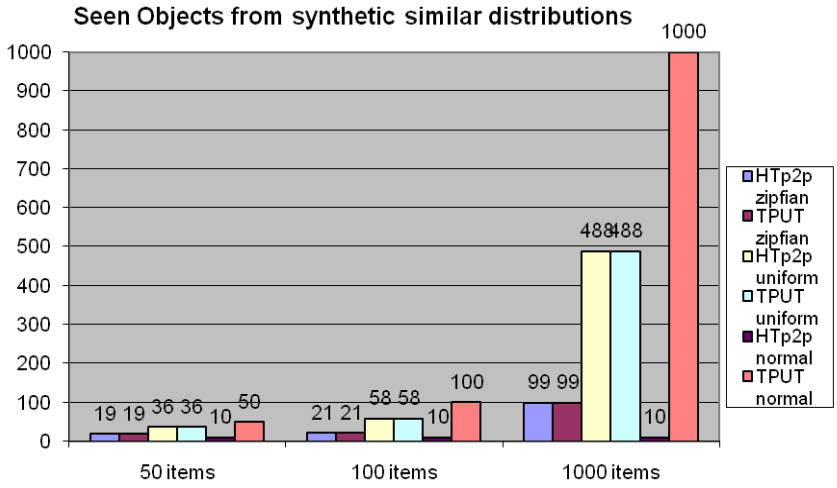

Fig. 16 Seen objects for various synthetic datasets

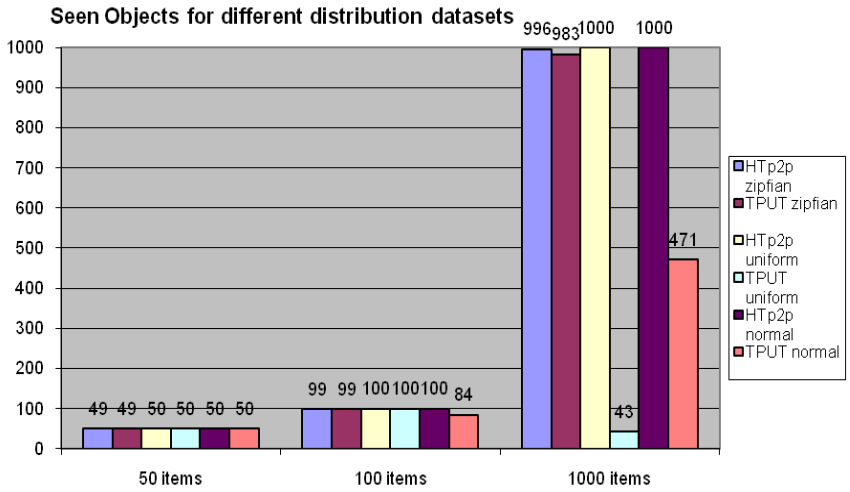

Fig. 17 Seen objects for various distribution datasets

5 Conclusion and Discussion

Top-k query processing is a of fundamental primary importance cornerstone for an effective search in distributed networks which is crucial for p2p applications such as digital libraries, social communities, preference queries over product catalogs, file sharing, sensor network monitoring, web logs analysis etc. Exact threshold-based top-k algorithms could be the solution for searching in modern p2p applications that maintain ranked data or take as input ranked data accordingly to their needs. The new extended versions of HT, HT-p2p and HT-p2p plus presented promising results in terms of bandwidth consumption which is the most crucial parameter as the size of p2p networks increases daily.

However, there are some points which need to be studied further in order to improve the time performance of threshold-based algorithms. Thus we could use heuristics to detect very large or low scores which could cause the recalculation of intermediate thresholds and to adapt relatively accordingly to our global thresholds. In addition, we could use different thresholds for each peer using techniques like these in [10] instead of global thresholds. As well, some further extensibility suggestions to the direction of speed improvement could be the caching of top-k results in order to return immediately the questions that have been answered in the early recent past or the application of timeout techniques for slow contributor peers or for delayed contributor super-peers. The last suggestion could enhance especially the switching from HT-p2p plus to HT-p2p to archive the speed up of the whole aggregation process.

Some other topics of discussion could be the definition of some criteria to define each time which peer could play the role of super-peer: accordingly to the computational power, to the network distance and topology, the possible network failures etc. The last constitutes to an important related problem of peer failures

during query execution. A solution to this problem can be found in [27], where a re-organization step for the affected portion of the query execution plan could be applied in each network failure. Also we could use of the notion of k-redundant Super-Peer which was introduced in [28]. A super-peer is k-redundant if there are k nodes sharing the super-peer load, forming a single "virtual" super-peer. Every peer in the virtual super-peer is a partner with equal responsibilities: each partner is connected to every client and has a full index of the clients' data, as well as the data of other partners. Peers send queries to each partner in a round-robin fashion; similarly, incoming queries from neighbours are distributed across partners equally. Hence, the query load on each partner is a factor of k less than on a single Super-Peer with no redundancy. Therefore, a k-redundant Super-Peer has much greater availability and reliability than a single super-peer and could be used to solve the specific problem of failure.

Apart from these extensibility suggestions our plans for the future include the comparison of the HT-plus in a new fully- distributed simulation platform to evaluate more precisely its behaviour with time. Also, we plan to study firstly the support top-k join queries and secondly the support of non- monotonic aggregation functions in order to enrich our query capabilities. These characteristics could give to the future peer data management systems the ability to of answering more complex queries and fulfilling more advanced user demands.

References

1. Fagin R.: Combining fuzzy information from multiple systems. In Proceedings of the 15th ACM Symposium on Principles of Database Systems, Montreal, (1996)
2. Chaudhuri, S., Gravano, L.: Evaluating top-k selection queries. In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99), (1999)
3. Fagin R. et. Al.: Optimal Aggregation Algorithms for Middleware. In Proceedings of the 20th PODS (2001)
4. Hayek, R., Raschia, G, Valduriez P., Mouaddib, N.,: Data Sharing in P2P Systems (2010)
5. Yu, H., Li, H., Wu, P., Agrawal, D., Abbadi, A. E.: Efficient Processing of Distributed Top-k Queries. In Proceedings of the 16th DEXA (2005)
6. Cao, P., Wang Z.: Efficient Top-k Query Calculation in Distributed Networks. In Proceedings of the 23rd PODC (2004)
7. Michel, S., Triantafillou, P., Weikum, G.: Klee: A Framework for Distributed Top-k Query Algorithms. In Proceedings of the 31st VLDB (2005)
8. Nejdl, W., Siberski W., Thaden U., Balke W-T.: Top-k Query Evaluation for Schema-Based Peer-to-Peer Networks. In Proceedings of the 3rd ISWC (2004)
9. Balke, W-T., Nejdl, W., Siberski W., Thaden, U.: Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. In Proceedings of the 21st ICDE (2005)
10. Neumann, T., Bender, M., Michel, S., Schenkel, R., Triantafillou, P.,Weikum, G.: Optimizing distributed top-k queries. In Proceedings of the WISE '08 (2008)
11. Neumann, T., Bender, M., Michel, S., Schenkel, R., Triantafillou, P.,Weikum, G.: Distributed top-k aggregation queries at large. Journal of "Distributed Parallel Databases '09", 26:3-27 (2009).

12. Xu, Y., Ishikawa, Y., Guan, G.: Effective Top-k Keyword Search in Relational Databases Considering Query Semantics. In Proceeding of the Asia-Pacific Web Conference (APWeb) and Web-Age Information Management (WAIM) '09 (2009)
13. Chrysakis, I., Plexousakis D.: Semantic Query Routing and Distributed Top-k Query Processing in P2P Networks. Technical Report. TR-387, ICS-FORTH, Greece (2006)
14. Tempich, C., Staab, S., Wranik, A.: REMINDIN': Semantic query routing in peer-to-peer networks based on social metaphors. In Proceedings of the 13th International WWW Conference, (2004)
15. Nejdl, W., Wolpers, M., WSiberski, W., Löser, A., Bruckhorst, I., Schlosser, M., Schmitz. C.: Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. In Proceedings of the 12th International WWW Conference (2003).
16. Hill, B.: A Lattice Framework for Reusing Top-k Query Results. In Proceeding of the IRI Conf. (2005)
17. Crespo, A., Garcia-Molina H.: Semantic Overlay Networks, for P2P Systems. Technical Report. Stanford University, USA (2003)
18. The JXTA Platform *http://www.jxta.org/*
19. S.Naicken, B.Livingston, A.Basu,S.Rodhetbhai, I.Wakeman, D.Chalmers.: The State of Peer-toPeer Simulators and Simulations. ACM SIGCOMM Computer Communication Review ,Volume 37, Number 2, April (2007)
20. The PlanetSim Simulator. *http://planet.urv.es/trac/planetsim/*
21. The Overlayweaver Simulator. *http://overlayweaver.sourceforge.net/*
22. The PeerSim Simulator. *http://peersim.sourceforge.net/*
23. The P2psim Simulator. *http://pdos.csail.mit.edu/p2psim/*
24. The SSFNET. *http://www.ssfnet.org/javadoc/*
25. The Internet Movie Database. *http://www.imdb.com/*
26. The Chord Project. *http://pdos.csail.mit.edu/chord/*
27. Akbarinia, R., Pacitti, E., Valduriez, P.: Reducing Network Traffic In Unstructured P2P Systems Using Top-k Queries. Distributed Parallel Databases 19(2–3), 67–86 (2006)
28. Yang, B., Garcia-Molina, H.: Designing A Super-peer network. In Proceedings of the 19th International Conference on Data Engineering (2003)