Semantic Query Routing and Distributed Top-k Query Processing in Peer-to-Peer Networks

Ioannis Chrysakis and Dimitris Plexousakis

Institute of Computer Science - FORTH

Vassilika Vouton, PO Box 1385, GR 71110, Heraklion, Greece and

Department of Computer Science, University of Crete

GR 71409, Heraklion, Greece

{hrysakis, dp}@ics.forth.gr

Technical Report ICS-FORTH/TR-387 – December 2006
Copyright 2006 by FORTH[±]

Keywords: {top-k queries, query processing, query routing, p2p networks, schema-based p2p systems}

Technical areas: {peer-to-peer systems, databases, semantic web}

Abstract

Requirements for widely distributed information systems supporting virtual organizations have given rise to a new category of peer-to-peer (p2p) systems called schema-based. In such systems each peer is a database management system in itself, exposing its own schema. In such a setting, a main objective is the efficient search across peer databases by processing each incoming query without overly consuming bandwidth. In this report, we adopt a super-peer-based architecture and suggest a query routing mechanism, upon which we propose a query processing technique for top-k queries. Top-k queries in the context of p2p systems give the opportunity to filter the results and to eliminate network traffic by choosing the k highest ranked results. We introduce HT-p2p and HT-p2p+, two extended versions of the Hybrid Threshold Algorithm adapted to our p2p scenario. For the evaluation of these algorithms we implemented a prototype system upon the JXTA platform. Extensive experiments with different data sets and parameters have shown promising results about the performance of the query processing strategy.

[±]Acknowledgement: Special thanks to Evi Dagalaki for her contribution to the evaluation process.

Introduction

Peer-to-peer (p2p) systems are distributed systems without centralized control in which each node shares and exchanges data across the network. The salient features of p2p systems make up a long list: low-cost sharing of data, redundant storage, self-organization, autonomy, load-balancing, fault tolerance. A new category of p2p systems, called schema-based, in which each peer is a database management system in itself and exposes its own schema, have recently drawn considerable attention. These systems can support distributed heterogeneous data stores and combine approaches from p2p research, as well as from the database and semantic web research areas. The combination of Semantic Web [1] and peer-to-peer technologies, i.e., the use of semantic descriptions of data sources stored by peers and of semantic descriptions of the peers themselves, is expected to improve query formulation in such a way that they can be understood by other peers, in merging the answers received from different peers, and in directing queries across the network.

However, such systems that broadcast all queries to all peers suffer from limited efficiency and scalability. Intelligent routing strategies are essential in such settings so that queries get only routed to a semantically chosen subset of peers able to answer parts of or whole queries. The problem of efficient query routing has been studied by many researchers since its solution affects the overall search mechanism of the p2p network. Moreover, in order to gain fast retrieval of data without large bandwidth consumption, there is a need for efficient ways of queries processing by a selected set of peers.

Top-k query processing has been studied recently as an attractive technique since p2p users in most cases are interested in a few most relevant answers to their queries. Simply put, top-k queries return only the k best results according to a given criterion. Generally, top-k queries on multidimensional datasets compute the k most relevant or interesting results to a partial-match query, based on similarity scores of attribute values with regard to elementary query conditions and a score aggregation function. By returning the k highest ranked results to each query a lot of processing and communication cost is saved across large scale information systems built upon a p2p network

In this report, after defining the query routing context and a suitable p2p architecture we propose a top-k query processing strategy. We introduce two versions of HT-p2p algorithm that extend the HT algorithm [2]. These extensions yield a number of advantages when compared with other approaches. The HT-p2p algorithm assumes that results are returned by executing an instance of the algorithm at a specified super-peer, named a Collector super-peer. HT-p2p+ assumes that results come from the combination of all top-k object sets that are returned from each running instance of the algorithm to each specified Contributor super-peer. For the evaluation of these algorithms we implemented a prototype system - called HT-p2p system - on the JXTA platform [3]. Extensive experiments with different data sets and parameters have shown promising results about the performance of the proposed query processing strategy.

Background and Related Work

Query routing in a p2p network is the process by which a query is routed to a number of relevant peers instead of being broadcast to the whole network. This problem has been studied in recent works. In [4] a routing strategy based on routing indices is presented. Obviously, indices only help if they can exploit and express regularities present in the peer and data distribution. A more advanced technique is presented in [5, 6] which introduce the notion of Semantic Overlay Clusters (SOCs) that define peer clusters according to the metadata description of peers and their contents. The former approach needs to accommodate index updates, whereas the latter needs an accurate definition of rules for peers joining into a SOC. Both of them use a super-peer topology. A super-peer is a node of the network that acts as a server to a subset of clients. This topology takes advantage the heterogeneity of peers; it is scalable as new peers join and seems to be most suitable for schema-based p2p networks since it can support heterogeneous schema-based systems with different metadata schemas and ontologies.

In [7], the INGA algorithm is presented. INGA extends the ideas of REMINDIN' [8], where each peer plays the role of a person in a social network. To determine the most appropriate peers, each peer maintains in a lazy manner a personal semantic shortcut index by analyzing the queries that are initiated by users of the p2p network and that happen to pass through the peer. The main limitation of this routing approach is the unavoidable flooding of the network with messages, when a new peer (that has not yet stored any shortcuts) enters the network or when peers (in lower layers) contain limited information about queries that have already been answered in the past. The SQPeer routing strategy [9] uses intentional active schemas (RVL Views) for determining relevant peer bases through the fragmentation of query patterns. However, since each view (active-schema) corresponds to a peer advertisement, it should be broadcast to the whole p2p network.

Query processing is the next step after the query routing task. Specifically, after the query is routed to a set of appropriate peers, query processing undertakes the task of combining the results from each peer and returning the final ones to the peer that submitted the original query. In this work, we focus on query processing for top-k queries. The state of the art in top-k query processing has been defined by the seminal work of Fagin et al on the Threshold Algorithm (TA) in [10]. In p2p networks, only a few works about top-k retrieval algorithms have been recently published. These are based on certain common assumptions. They assume that each peer (source) has maintained a list of pairs: (Object_id, score), where Object_id is unique. Also, each pair is related with one attribute (property). Almost all approaches assume that these pairs pre-exist. There is only one approach [11] that deals with the ranking and the use of ranking methods (Topic-distances in Taxonomies, TFxIDF). Finally, all these top-k query processing approaches support only selection queries. We classify these approaches into four categories.

The first category - termed the "*Probabilistic – Histograms Approach" - belongs* to a family of algorithms [12, 13, 14, 15] that 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 data structures like routing filters and histograms. The main limitation of this approach is that it falls back to broadcasting when the desired number of results is too high or, when the user asks for a high degree of accuracy. Also all algorithms of this family that are based on the TA need several round-trips in order to retrieve the final results.

The approach of *Nejdl. et. al.* [11] relies on a super-peer backbone organized in the Hypercup [16] topology upon a RDF-Based p2p network. They combine ideas of semantic query routing based on indices [4] and propose a decentralized top-k query evaluation algorithm for p2p networks which is based on dynamically collected statistics put into local indices. A more sophisticated top-k processing framework is introduced in [17] but it still relies on such statistics. Thus, the first time all peers have to participate in processing the query. Then several round-trips are required in order to retrieve the final result. This often leads to situations where peers have to wait for each other. Also, in the case where the query is not contained in indices the algorithm becomes more time and network consuming.

Bruno, Marian et. al. introduced the *Upper Algorithm* [18], an algorithm for evaluating top-k queries over web-accessible databases. The Upper Strategy selects a pair (object, source) to probe next based on the property that the object with the highest upper bound will be probed before top-k solution is reached. [19] presents the *pUpper Algorithm*, an improved version of Upper which enables parallel top-k processing and focuses on reducing query response time through the use of queues in order to gain the lost time from the delay of accesses at each peer (source). One of the significant advantages of this approach is that the top-k query processing strategy doesn't require complete knowledge about the scores at each step. However, the main drawback of this strategy is that only one source can be accessed at a time. This could be restrictive if we consider a p2p network, where a large number of peers must wait for the others at each step of the algorithm to access the suitable (Object, Score) pairs.

Another approach to top-k query processing, termed the *"Three Phase Threshold Approach"*, comprises algorithms (TPUT, TPAT, TPOR, HT) that consist of 3 basic phases and use thresholds in order to return the top-k results [20, 2]. These algorithms aim at answering top-k queries over large-scale networks efficiently. Although they belong to the class of Threshold Algorithms, they overcome the problems of TA like unbounded message rounds. TPUT outperforms TA in most cases but it unrealistically assumes uniform data distribution among nodes [20].

The *Hybrid-Threshold algorithm (HT)*, combines the advantages of both TPUT and TPOR, and demonstrates that it is robust to different data distributions as its evaluation was shown [2]. It is based on partial sums and upper bounds to prune non-eligible objects, and terminates in a fixed number of 3 + 1 phases. Finally, HT fits well in a 2-tier distributed system and estimates data distributions without a-priori knowledge, which means that it could work well even if the number of peers is limited and the knowledge about their data is limited. Based on these observations we opted to extend the HT algorithm under our peer-to peer scenario.

Methodology

Basic Context and Directions

Assume a p2p network in which each peer hosts and maintains a database (db). We can view each peer as a person or group that share their knowledge with other people or groups. It is desirable that each person or group have the ability to reuse others' knowledge at least at the database schema level. Our main goal is the efficient search across the p2p network whilst exploiting the easy sharing of databases. For the accomplishment of this goal it is crucial that each query is not broadcast into the whole network, but is routed to a selected set of relevant peers. Furthermore, the efficiency and good performance of the whole p2p network does not only depend on how the query is routed to relevant peers, but also on how it is processed by these relevant peers.

To support peer autonomy and network self-organization, we choose an unstructured p2p environment, since *unstructured networks* can provide this property and may additionally support rich query languages in contrast to structured DHT networks. Regarding network topology, we adopt a super-peer architecture. *Super-peer networks* combine the efficiency of centralized search (super-peers route queries to appropriate peers) with the autonomy, load balancing and robustness provided by distributed search. As long as super-peers receive the messages and these are not broadcast across the network, we have an efficient usage of network bandwidth by limiting traffic while we can take advantage of the heterogeneity of capabilities (e.g., processing power) across peers and can cluster them according to defined criteria. Furthermore, super-peer based networks can provide better scalability than pure p2p networks and eliminate the phenomenon of bottlenecks which can potentially occur for broadcast-based networks (pure P2P).

Given that each peer is managing a database, our p2p network fits well with the idea of *schema-based P2P networks*. In addition, our suggested super-peer topology can provide support for heterogeneous schema-based networks with different metadata schemas and ontologies. For this reason we take advantage of standards such as XML [21] and RDF/S [22] and build a p2p system that exchanges semantically associated information across the network. We assume that schemas and/or ontologies are formulated in RDF/S in order to be reusable and extensible. In conclusion, we advocate an unstructured RDF Schema-based p2p network under a super-peer topology. These considerations are taken into account in our query routing and processing strategy.

Query Routing Strategy

Ontologies are a key enabling technology for Semantic Web applications. In a simple form ontologies are *taxonomies* representing concepts and relations. We assume that each database exposes its own *taxonomy of terms* that describe its contents at the schema level. Each peer follows the model of [23] and can be thought of as a simple source. A simple source consists of a taxonomy and an object base that indexes objects under the terms of the taxonomy. Terms are connected through *isA links*. In our case, each peer is a source with a corresponding taxonomy indexing the database containing the data. We assume that each peer has its own RDF/S schema describing its database schema information and includes the corresponding taxonomy of terms related to the specific database and other user-defined relationships and properties.

Furthermore, we assume that there are bidirectional links (cross links) between terms of different taxonomies. In practice, each peer can make an "in relation" isA link by linking a term of its taxonomy with a term of another peer. In this way, each peer not only manages its specific schema and data, but it can enrich them by using other related terms. Figure 1 depicts an example of two taxonomies of terms belonging to distinct peers and associated with bidirectional links.

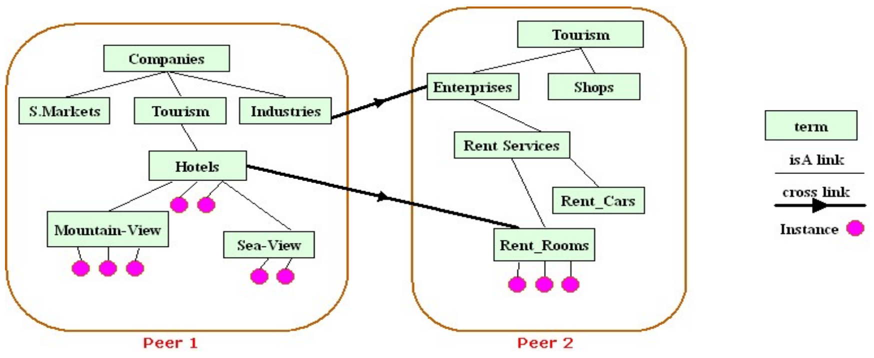

Fig. 1. An example of two taxonomies with cross links

The clustering of peers according to semantic criteria would enable a more accurate routing strategy in the process of discovering the relevant peers to a specified query. *Semantic Overlay Networks (SONs)* [24] provide for an intuitive way to cluster together peers sharing the same schema about a community domain. We assume that each super-peer is joined to at least one specified SON and is responsible for it. We can use a clustering policy similar to the approach based on *Semantic Overlay Clusters* [6]. We assume that each super-peer deals with certain subject matters or topics that collectively characterize its SONs.

For querying RDF/S schemas we need an expressive query language that exploits schema structure. *RQL* [25] is a typed language which supports generalized path expressions with variables on labels for nodes (classes) and edges (properties). We choose RQL as it provides complete query functionality in comparison to other RDF query languages (RDQL, Triple, SeRQL, Versa, N3), ([26], [27]) and has the advantage of disallowing cycles in subsumption hierarchies. The latter is of crucial importance as we have isA links among different peers.

When a new peer requests to join the SON of a specific super-peer, the latter applies the defined clustering policy to accept it in its cluster. In this way semantically irrelevant peers cannot be joined in the same SON, and thus they cannot be in the specific cluster of the super-peer. One clustering policy that can be used in our context is the matching of the candidate peer's taxonomy terms with those of the super-peer. The initial role of super-peers is to collect the RDF/S schemas of the peers in their clusters.

The Query Routing Algorithm

Let's assume a number of super-peers that have in their responsibility a cluster of peers according to their corresponding SONs. For simplicity let's assume that only one SON is in each super-peer's responsibility (see an example in Figure 2 below). Assuming that peer P1 (which belongs to SON1 and its responsible super-peer is Super-Peer1) issues a query Q, the *query routing algorithm* proceeds as follows:

- We first find the responsible super-peer for P1 which in this example is Super-Peer1.
- Super-Peer1 examines the RDF/S schemas of the peers in its cluster and finds all the relevant peers according to the matching of their terms (at schema level) and their properties (at instance level) with the query predicates.
- If Super-Peer1 finds relations between peers in different clusters (i.e., through the use of isA cross-links), it adds to the set of relevant peers the ones that match the query predicates
- Then the final set of relevant peers and their corresponding super-peers are returned.

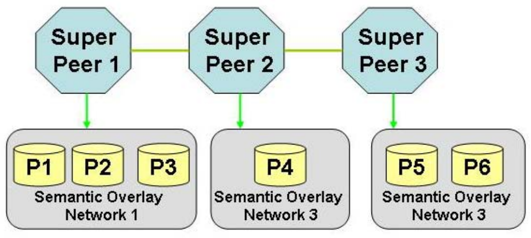
Super Peer 1
Super Peer 2
Super Peer 3
P1
P2
P3
Semantic Overlay Network 1
P4
Semantic Overlay Network 3
P5
P6
Semantic Overlay Network 3

Fig. 2. Super-peers and Semantic Overlay Networks

We should point out that the definition of "relevant peers" can be adapted to the needs of the applied p2p network. Surely a relevant peer must match at schema-level with at least one property and one relation of a specified term. The strict definition corresponds to a case where all properties and relations (at schema-level) are defined on the peer's RDF/S schema and the requested conditions on their values are valid. Also, we can conclude that super-peers play an important role in the suggested routing process. Peers are grouped based on the semantics of their stored data. Thus, since queries are routed according to the same semantics-based classification policy, required results to each query are found faster and are contributed only by the relevant to the query peers.

Furthermore, the flexible use of taxonomies, their corresponding RDF/S schemas and the links among them facilitate the routing process which does not need to apply complicated and time-consuming tasks such as index maintenance and updates. The bidirectional links are useful when we are looking for a term that does not exist at a peer, but this peer "knows" that one of its terms is related with the asked term but is contained in another peer. Hence, each query is routed to a set of relevant peers only with low bandwidth consumption across the p2p network.

Advantages of Query Routing Strategy

In general if we have an RDF/S Schema to describe each peer's knowledge base (data) then our routing technique can be applied to heterogeneous databases. The only thing, we should need to support in order to get back the results, is a kind of wrapper that would undertake the transformation of the query to a suitable query language for each database in order to proceed the matching process. In addition, for Semantic Web this technique is surely applicable since everything can be thought as a RDF description and there is interoperability between the common standards (XML [21], RDF/S [22], OWL [28]).

We can obviously conclude that we abuse the network topology, that's why the Super-Peers have an important role to all the suggested routing process. Thus, a meaningful advantage of this query routing context is that peers are grouped based on the semantics of their stored data. Thus, since queries are routed according to the same semantically-based classification policy, required results to each query are found faster and only from relevant to the query peers which are considered. We should mark that peers that have few or no results considering a given query will not be contacted, since their classification will assign them to different Semantic Overlay Networks, thus avoiding wasting processing and communication resources on that requests.

Furthermore, the flexible use of taxonomies, their corresponding RDF/S schemas and the links among them facilitate the routing process which does not have to take care of complicated and probably time-consuming tasks such as index maintenance and updates. The two-way links obviously helps in the case when we are looking for a term that does not exist at a peer, but this peer "knows" from a two-way link that one of its terms is related with the asked term but is contained to another peer.

Finally the harmonic combination of the building blocks of our query routing context provides a routing strategy that supports self organization and distribution of peers, simple, accurate and not complicated procedural steps. Therefore each query is routed only to a set of relevant peers with low bandwidth consumption across the applied p2p network where the query will be processed by the proposed processing strategy which is presented in the next section.

Top-k Query Processing Strategy

General Issues, Basic Features and the HT-p2p Context

Bearing in mind that p2p systems are designed to support global-scale information systems, it is easy for a user to obtain a huge amount of results in response to a query. Thus, it is important to support a top-k query processing strategy in order to filter results, accommodate partial matches and contribute to a good overall performance of the p2p system. Also, top-k query processing is undoubtedly relevant in the context of P2P systems since the p2p users are in most cases interested in a few most relevant answers to their query. In general a top-k retrieval algorithm returns the"best" k results according to a given criterion.

First, the top-k query is issued by one participant peer. The query routing algorithm takes on to find the peers relevant to the query and forwards the query to them. The relevant peers apply a scoring technique upon their data, and all the candidate top-k objects are returned along with their scores. Specifically, (Object_id, Score) pairs are returned as input to the top-k processing algorithm. Finally the last returns the top-k objects and their required data from the peers where they are located to the peer which originates the query.

The algorithm *HT-p2p* is an extended version of the HT algorithm [2] adapted to our p2p setting. It can be applied in any super-peer based p2p network as an efficient top-k processing algorithm. HT assumes that there are m nodes and one single central manager in a Content Distribution Network (CDN) [29]. In HT-p2p we assume that a large number of super-peers are responsible for a larger number of peers. In particular, each super-peer has a cluster of peers (SON). Let's assume that we have m peers at each cluster. In HT-p2p each participant super-peer saves intermediate results which helps pruning some steps of the basic algorithm.

Before describing in detail all the required steps for HT-p2p we should explicitly define the roles that peers play during the execution of the algorithm. The peer that issues the original query across our p2p network is called an *Originator peer*. Respectively, its responsible super-peer plays the role of *Originator super-peer*. Each peer can be a *Contributor* or a *Non-contributor peer*. The first participates to the execution of HT-p2p and contributes to the top-k results. This should be a relevant peer to the query as returned from the query routing strategy. A Contributor peer sends the required (Object, Score) pairs to the specified Super-peer. A Non-contributor peer at the specific running instance of HT-p2p does not participate and is in practice inactive. A *Collector super-peer* executes the specific running instance of HT-p2p. It collects the intermediate results from all the contributor peers and finally returns them to the Originator super-peer. Each time a query is processed there is only one Originator peer and its corresponding *Originator super-peer*. The collector super-peer starts and runs the specific instance of HT-p2p. It could be the Originator super-peer, or any other. 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. At each running instance of HT-p2p all the Contributor peers participate as determined by the query routing strategy that detects the relevant peers to the top-k query.

For HT-p2p we use the same query model, as in the "Three Phase Threshold" Approach [2, 20]. Therefore 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 object. The objects 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. Each specified super-peer issues a top-k query and retrieves objects from the network with the k highest value of f(S1(O),...,Sm(O)), where f is a monotonic function, to compute the overall score of an object. L We should note at this point that each object is scored according to the selected scoring technique which in turn determines the applied monotonic function. We use the SUM function for ease of exposition. Objects can be thought of as RDF resources if peers host RDF/S data or tuples if they host relational databases.

The HT-p2p Algorithm

After the set of ranked objects of relevant (contributor) peers across the p2p network are returned, an instance of HT-p2p is run by the Collector super-peer. 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 the intermediate results of this phase (seen objects, their scores, and their partial sums).

■ For an object O, the *partial sum* $S_{psum}(O) = S'_1(O) + \ldots + S'_m(O)$ where $S'_i(O) = S_i(O)$ 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 one time, so it has been stored.

Phase 2:

■ The Collector super-peer broadcasts 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"*: is the k-th highest partial sum.
- m = the number of peers at the specified cluster of super-peer.

■ Upon receiving the list L, for each object O_j in L: peer i finds its local score V_{ij} and determines the lowest local score S_lowest(i) among all the k objects in L. If O_j does not occur in the local list then $V_{ij} = 0$.

■ Then peer i sends the list of local objects whose values are greater than or equal to Ti, where Ti=max(S_lowest(i),T), to the collector Super-peer.

■ 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 Originator peer which made the original top-k query.
- But if (Ti > Tpatch), then two additional phases (Phases 3 and 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, and their partial sums).

Phase 3 (patch phase if 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.
- Then the super-peer tries to prune away more objects by calculating the upper bounds of the objects seen so far and have been stored till now.
- An *upper bound* for an object O (U_{sum}(O))is calculated by the formula: Usum(O)= S'_1(O) + S'_2(O), + … S'_m(O), where:
    - S'_i(O) = S_i(O) if O has been reported by node i; S'_i(O) = min(Ti, Tpatch) otherwise.
- Then the super-peer removes any object Oj from the candidate set whose upper bound is less than τ3 and return the top-k candidate set.

Phase 4 (necessary if we run Phase 3):

- Since the Collector super-peer stores the intermediate results (seen objects, their corresponding scores and partial sums of them), at 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 identify the exact top-k objects.
- Finally it sends the top-k objects to the Originator super-peer which in turn *returns* them to Originator peer which issued the original top-k query.

HT-p2p is an extended and improved version of the Hybrid Threshold algorithm [2] adapted under our p2p scenario. Its correctness is based on the proof of HT [2]. The advantage of HT-p2p is that it can return - when no patch phase is needed- the final results in phase 2 because of the ability to store intermediate results. For the same reason, phase 4 of HT-p2p, as compared with phase 4 of HT, does not need to request peers to send their scores since they have been saved at the previous phase. Furthermore, all Contributor peers can execute in parallel except from the case where the calculation of phase-2 bottom is needed for determining Tpatch. At this specific point peers should wait until *phase 2 bottom* is determined.

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. Then we should combine the results from all super-peers and calculate the real scores for their "top-k" objects in order to find the k-highest ones which denote the final top-k results. The last process requires super-peers to receive scores from their corresponding peers that maybe were not sent at the real execution of the specific HT-p2p's running instance. This approach introduces a modified version of HT-p2p that we call it *HT-p2p+*.

We can observe that by applying HT-p2p+ we have a more distributed processing policy which promises better performance when we have a large number of relevant peers at different super-peers. Also, potential bottlenecks to a single Collector super-peer are eliminated by using multiple instances of the algorithm which contributes to the sharing of the communication load at Contributor Super-Peers. On the other hand, for each running instance we have extra processing and probably communication cost (calculation of real scores of unseen objects) in order to finally combine the results at the Collector super-peer. Obviously there is a trade-off between performance and network consumption which is a common issue at distributed environments like p2p networks. In the next subsection we elaborate on the points in which HT-p2p+ differs from HT-p2p.

The HT-p2p+ Algorithm

In HT-p2p+, an additional role has to be defined for super-peers, namely the role of *Contributor super-peer*, which contributes to the final top-k results by applying a running instance of HT-p2p across its relevant peers. Similarly to the definition of Non-contributor peers, the *Non-contributors super-peers* don't run any instance of HT-p2p since they don't have any relevant peers. Specifically, as the routing strategy has returned the set of ranked objects of relevant (Contributor) peers for each corresponding (Contributor) super-peer, an instance of HT-p2p+ 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 anyone of the Contributor super-peers. The role of Collector super-peer in HT-p2p+ is to collect all the top-k results from all the running instances of HT-p2p+, combines them and finally returns them to the Originator super-peer.

Except from these roles, HT-p2p+ differs from HT-p2p in some of the executed steps. In all phases the active role of super-peer is that of Contributor rather than Collector. Thus the Contributor super-peer interacts with the contributor peers at each phase and stores all the intermediate results of this phase (seen objects, their scores, and their partial sums). The Collector super-peer acts at the end of phase 2 (if we don't have a patch phase) and at the end of phase 4 of HT-p2p+. In these two cases, the specific Contributor super-peer sends the results to Collector super-peer. Then, the Collector super-peer combines all the results from the Contributor super-peers and returns the top-k objects to the Originator super-peer.

We should analyze further how the combination of results takes place, as it may require an *extra phase*. Assuming that each Contributor super-peer has sent its top-k object set to the Collector super-peer, the latter must choose the objects with the k-highest real aggregated scores among all top-k object sets. Thereby, it accumulates all the distinct candidate objects and sums their real scores. The Collector super-peer may ask each Contributor super-peer to calculate the real score for each object if it is not contained in its top-k object set. In this case all contributor peers need to send their scores to the Contributor super-peer (as long as they have not sent them during the execution of HT-p2p+) in order to calculate the real score for each object. The objects with the k-highest real scores are the final top-k objects.

Data Scoring and Use Cases of HT-p2p

According to the scoring technique and based on our suggested routing strategy we can define and specify more than one use case of HT-p2p. We have assumed that each query across the peer-to-peer network is a top-k query, so we look for top-k objects which have the k highest overall scores. HT-p2p takes as input sorted lists of (Object, Score) pairs. There is one list for each contributor peer. If we consider a peer-to-peer network where each peer is autonomic to rank its objects, then the same object would probably has different scores at different peers. We can think an example of a scenario where peers are ranking movies according to their preferences and we are looking for the movies with k highest overall score. In this case a scoring function could be a monotonic function (such us SUM).

Another scenario could be the case, where we have a top-k query upon some attributes, and the contributor peers don't have information about all attributes. Let's assume that we are looking for the top-k hotels that have: price < 100 Euros and rating>3 stars and distance<5 km from general hospital. In this case price, rating and distance are the specific attributes of the top-k query. Then some peers maybe do not have in their database information about an attribute, for example distance from general hospital. Our scoring technique could put zero in this attribute and could sum the others attributes to compute the final score at each peer. Furthermore we could use a weighted monotonic function for the computation of the final score of each object: *Final Score = w1*s1 + w2*s2 + w3*s3* where w1,w2,w3 are the predefined weights for each attribute according to the preferences of originator peer. Since the attributes are numerical we could denote that their corresponding scores s1, s2, s3 are ranked according to how close are their values to these that are requested at the original top-k query.

The last and perhaps the trivial and simple one is the case where each peer ranks the same objects with the same score by applying the same scoring function. Then a top-k query has meaning of use if peers don't contain exactly the same objects. Because in the case where all peers have the same objects with the same scores, we don't need to make a top-k query, but retrieve the highest k scores just from one peer. At the specified simple case, we could use any kind of monotonic function to rank objects at each peer. HT-p2p will return in this case as well the top-k objects with the highest values of the monotonic function across the peer-to-peer network.

Cost analysis of HT-p2p/HT-p2p+ Algorithm

Since the scalability and the general performance of a peer-to-peer network depends upon the communication costs among peers, it is important to consider a cost analysis for HT-p2p/HT-p2p+. In each phase we should examine the number of standard messages that are exchanged between peers and super-peers. Assuming that we have w Contributor super-peers (case of HT-p2p+), m Contributor peers at each cluster, n peers that have to run patch phase (n<m) and the Collector super-peer is contributor as well, we show the required message exchanges at each phase in Table 1 below.

Phase	Sender	Number of messages
Phase 1	Contributor peer(s)	m * w
Phase 2	Contributor super-peer(s)	w * m
Phase 2	Contributor peer(s)	m * w
Phase 3	Contributor super-peer(s)	w * n
Phase 3	Contributor peer(s)	n * w
Phase 4	Contributor super-peer(s)	1
Phase 4	Contributor super-peer(s)	1

Table 1. Required messages for a complete execution scenario of HT-p2p+

We should note that, if we run just one instance across the p2p network (case of HT-p2p), then we don't have any Contributor super-peers but only one Collector peer that does the entire work. In fact in this case our contributor super-peer is the Collector, hence w = 1. The final number of required messages for this completed scenario to transfer across the p2p network is determined as follows: *Total Number of Messages in HT-p2p+ = 3 m * w + 2 n * w + 2. Total Number of Messages in HT-p2p = 3 m + 2 n + 2.*

Applying HT-p2p: an example

With reference to Figure 2, 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 peer3, so SP1 is also the Originator super-peer and we are looking for a top-2 query (k = 2). Table 2 below depicts the (Object, Score) pairs at each peer of SP1. The execution of HT-p2p has to return the top-2 objects with the highest overall scores without examining all objects of each sorted lists at each peer in standard processing steps. The steps performed by HT-p2p for the sample top-2 query are the following:

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)		

Table 2. (Object, Score) pairs at each peer of SP1

Phase 1: 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 (S_{psum}) for all seen objects: S_{psum}(O4) = 21, S_{psum}(O2) = 17, S_{psum}(O5) = 48, S_{psum}(O1) = 29, S_{psum}(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.

Phase 2: 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. Then each peer first determines its lowest score on the objects of the list L (O5, O3). 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). Then, each peer calculates its threshold Ti = max(S_lowest(i),T) and sends its objects whose values are greater than Ti to SP1. For peer1, T1=11 so it sends objects (O4, O2, O5, O3) to SP1 with their scores, for peer2 T2=26, so it sends objects (O5, O1, O18, O3) with their scores and for peer3 T3=14 so to SP1 are sent only the pairs (O3, 30), (O5, 14). Afterwards SP1 calculates the partial sums for all objects seen so far and identifies the objects with the k highest partial sums. The partial sums are: S_{psum}(O4) = 21, S_{psum}(O2) = 17, S_{psum}(O5) = 57, S_{psum}(O3) = 67, S_{psum}(O1) = 29, S_{psum}(O18) = 29. Thus, the 2 highest are S_{psum}(O3) and S_{psum}(O4) where the last is equal with τ2 since it is the kth. Therefore Tpatch = τ2 / m = 57 / 3 = 19. Now SP1 checks if there is a need for patch phase at each peer by checking the condition Ti > Tpatch. For peer1 and peer3 there is no such need 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.

Phase 3: SP1 requests from peer2 to send all its objects that are no less than Tpatch = 19. Thus peer2 sends {O4, O14, O16, O13}. 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. For these objects SP1 calculate their upper bounds at peer2: 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. Then 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.

Phase 4: 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 peer3 (Originator peer) as top-k objects.

Advantages of Top-k Query Processing Strategy

Our top-k query evaluation strategy (including both versions of HT-p2p) takes as input (Object_id, Score) pairs which are sorted in descending order. These pairs are returned by the scoring technique which takes as input the objects from relevant (contributor) peers that are returned by the query routing algorithm. Hence, our query processing strategy collaborates with our query routing strategy, but it is independent of it. In other words any query routing technique could be applied in our setting in order to define the top-k candidate objects. Table 3 compares HT-p2p with dominant approaches in top-k query processing for distributed networks. HT-p2p and the approach of [11, 17], are the only ones that are adapted to super-peer p2p networks and, as such, take advantage of the heterogeneity of peers and of a topology that reduces bandwidth by eliminating the transferred messages between peers. All approaches are distributed in some ways, because some or all peers execute specific steps in the respective algorithms. In HT-p2p, each peer has a discrete role in the processing strategy and at each phase each contributor peer runs independently of each other and in a distributed manner.

	HT-p2p / HT-p2p+	Marian	Nejdl	Probabilistic
Adapted to Super-peer P2P	Yes / Yes	No	Yes	No
Distributed Algorithm	Yes / Yes (Fully)	Yes	Yes	Yes
Delay at peers	In Phase 2 / No	In each phase.	When there is not enough index information.	Trade-off between quality and performance.
Standard Phases	Yes (2 + 2) / Yes (2+2+1) for each Contributor super-peer	No	No	Several round-trips because of TA-Style algorithms.
Estimation of scoring distribution	Without a-priori knowledge. / Without a-priori knowledge.	Without a-priori knowledge.	Not addressed.	Based on probabilities. Assumes each peer owns others' histograms.

Table 3. Comparison between top-k query processing approaches

Indeed, in Phase 2 of HT-p2p there is a delay cost where all the Contributor peers are waiting for a universal threshold. This problem is avoided in HT-p2p+. The approach of [19] enables parallel top-k processing but focuses only on reducing query response time through the use of queues. In [11, 17], each peer computes local rankings for a given query, results are merged and ranked again at the super-peers and routed back to the query originator. According to the hypotheses of each approximate algorithm, each peer participates to the calculation of suitable probabilistic guarantees that contribute to the final pruning of top-k candidate objects. Thus, the delay at peers appears in HT-p2p only in phase2, while in [18, 19] it appears in each phase since one source (peer) can be accessed at a time in each phase of the algorithm. In [11, 17], a delay is incurred because information for the specified query must be collected from peers. In approximate top-k retrieval algorithms that belong to the probabilistic approach delays are introduced at peers when the desired number of results is too high or when the user asks for a high degree of accuracy.

Another advantage of our proposed approach is the fixed number of standard phases in HT-p2p/+. This is not the case for the other approaches. Specifically, in the probabilistic approach there is a large number of round trips because the corresponding algorithms are based on TA [10] which does not take into account scoring distribution and works until it finds the k objects whose aggregated scores are no less than the current suitable threshold. Finally, our approach does not presuppose knowledge of score distributions. This characteristic is also supported in [18, 19] and generally leads to a fully dynamic strategy, which works well even if the number of peers is small and the knowledge about them is limited. In the probabilistic approach the estimation is based on specific probabilities (guarantees) which are calculated from the specified histograms that are assumed to be found in each peer for its neighbors.

Implementation

The evaluation of this work is focused on validating the proposed top-k query processing strategy under realistic scenarios. For this reason we implemented a system which invokes both versions of HT-p2p using the JXTA platform [3]. JXTA provides a common set of open protocols and an open source reference implementation for developing general purpose, interoperable and large scale P2P applications. By selecting the Java Version of JXTA we benefit from the application of both technologies. With Java our system is platform independent, with JXTA is network independent. Moreover since messages on JXTA platform are XML-based, we gain application independence by using this standard. XML supports interoperability across different applications, and in our case these applications could be different kind of peers (computers, PDAs, mobile phones etc).

The HT-p2p system as we called it, uses a variety of features from JXTA technology such as peers and peergroups, bidirectional pipes (*JxtaBidiPipes, JxtaServerPipes*), advertisements (*PeerAdvertisments, PipeAdvertisements).* It is built upon a super-peer topology. Each peer is assumed to have a ranked list of object-score pairs. We assume that each Object_id is unique and is related with a specific object. At each execution of HT-p2p the participant peers and super-peer(s) are exchanging messages that include (Object_id, Score) pairs, thresholds and some control information such as their peer name. The HT-p2p's architecture consists of two modules according to the type of participant peer, namely the *Super-Peer Module* and the *Peer Module*. Both define and invoke the required methods. Each module communicates with each other using a third abstract module called the *Communication Module*.

We should point out that the Super-Peer Module is designed using a *multi-threaded architecture* in order for each super-peer to communicate and to process each peer's request independently of the others at each phase of the algorithm. In general, the running of multiple threads in an application at the same time performs different tasks at this application [30] As long as peers are equal; we adjust all threads to have the same priority in order to be processed equally from the specified Super-Peer. Thus if for example (Figure 3) peer1 sends their (Object_id, Score) pairs in a request and peer2 its own pairs in another request, then the Super-Peer will process them independently. Therefore, at the same time and it could send back some results in parallel to each peer, since for each request a new thread is made to serve it.

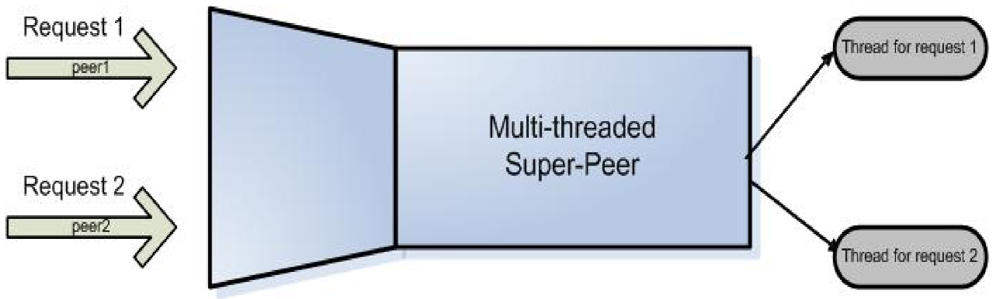

Fig. 3. Multi-Threaded Architecture of Super-Peer

With the HT-p2p system we proceed to extensive experiments under realistic scenarios and different parameters. Thus we examined in practise how HT-p2p algorithm behaves in a Super-Peer based peer-to-peer network under different conditions. We should mention at this point that for evaluation reasons, we modify the system in order to have an auto-configuration of peers, since the JXTA platform requires a unique port and a specific peer id. Thus each participant peer, as it joins the peer-to-peer network it gets a specific id (peer name) and an identifier (peer id). (e.g. peer_1, port: 9701). These elements are unique for each peer and they are used in all the communication and execution processes.

Evaluation

Experimental Setup

For all experiments in the HT-p2p system we used the Apache Ant version 1.6.5 [31] in order to run a cluster of peers at once. Each peer which is a java file runs in parallel with the rest ones through the use of this Java-based build tool. The super-peer and each cluster of peers (5-10) were run on a Pentium4 3.4 GHZ computer with 1.99 GB of RAM, Windows XP OS and were connected on a 100 Mbps LAN. We also, used Java version 1.5.0_08-b03 and JXTA version 2.3.6. For all experiments we used at least two kinds of scoring distributions: zipfian and normal. For zipfian distribution the zipfian parameter was 1 and the range of values was [1, 500] while the seed was different for each peer in order to get different values. For normal distribution the mean was 150 and the standard deviation was equal with the id of each peer in order to vary the scores across peers. In one experiment (3) we used additionally the uniform distribution for range [1, 500]. For the normal and uniform distribution generators, we used the SSF net package [32]. We assumed for Super-Peers that they are online and they have created (read) their corresponding PipeAdvertisement. For peers we also assume that they are online, they have read the PipeAdvertisement of each corresponding Super-Peer so they joined in each peer group. This assumption was necessary in order to start measuring the clear execution time of HT-p2p without taking into account the time that the peers need to join into the peer-to-peer network.

Experiment 1

The first experiment was done to evaluate HT-p2p behaviour as the number of contributor peers increases. We measured the execution time of the algorithm after all peers had joined the p2p network, the number of top-k candidate objects (final objects), the number of seen objects and the patch phases that run during the execution of the algorithm. We should mention that we put a sleep statement for each peer's first request, in order to wait until all peers have entered the p2p network and from this point to start measuring the execution time. This statement proved necessary when the number of contributor peers exceeded 10, in order for these peers to connect simultaneously into the JXTA server pipe. It was necessary just for evaluation reasons, because phase 1 of the algorithm could start before all peers have joined in. We measured execution times with and without the sleep statement. Each peer had 150 objects and their corresponding scores were following the normal or the zipfian distribution. We run the same cases at least

3 times for all top-10 queries that we processed. Table 4 depicts the results of this experiment and Figure 4 summarizes graphically the full case where sleep statement was enabled.

Peers	**Distribution**	**Seen Objects [Final Objects]**	**Patch Phases**	**Time (ms) [Average] Sleep Statement Enabled**	**Time (ms) [Average] Sleep Statement Disabled**
5	normal	150 [150]	0	17789,18563, 19751 **[18701]**	3719, 3831, 4113, **[3911]**
5	zipfian	100 [19]	1	15750 , 16296, 16938 **[16328]**	2094, 2125, 2359 **[2192]**
10	normal	150 [150]	0	19141, 21969, 22828 **[21312]**	12453, 13281, 14253 **[13329]**
10	zipfian	128 [128]	0	17595, 18235, 19124 **[18318]**	8092, 9424, 9922 **[9146]**
15	normal	150 [150]	0	30813, 33891, 45548 **[36750]**	
15	zipfian	140 [26]	1	29485, 31344, 42141 **[34323]**	
20	normal	150 [150]	0	33422, 48688 58703 **[46937]**	
20	zipfian	142 [28]	1	21954, 47000, 57469 **[42141]**	
25	normal	150 [150]	0	33485, 42626, 74532 **[50214]**	
25	zipfian	143 [143]	0	35673, 36375, 39922 **[37323]**	
30	normal	150 [150]	0	61407,115486, 131205 **[102709]**	
30	zipfian	144 [144]	0	46220, 73658, 119189 **[79689]**	
35	normal	150 [10]	23	155720,169674, 187987 **[171127]**	
35	zipfian	144 [144]	0	100495, 129017 209362 **[146291]**	

Table 4. Results of experiment 1

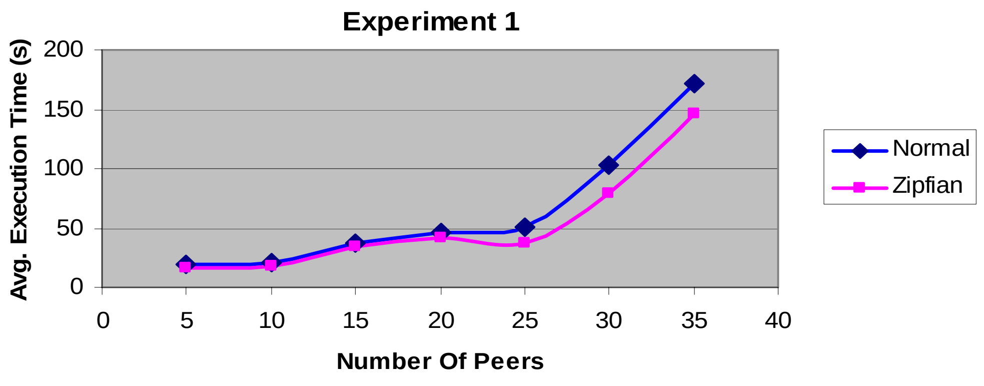

Fig. 4. Average time as the number of peers increases and sleep statement was enabled (exp.1)

The first observation is that the sleep statement causes a delay that affects the overall execution time. Specifically this delay for small number of peers proved even bigger than the total execution time. For this reason the differences in execution times between 5 and 10 peers are very small. Another significant observation is the fluctuations (smaller when the number of peers was up to 10 and larger when was above 10) on the execution times. It is worth noting that there are cases when the number of peers is larger (e.g. 25) while the execution time is smaller comparing with cases of fewer peers (e.g. 20). Our possible explanation on the noticed fluctuations could be the distributed running of algorithm's phases. This relates with the running of phase 2 at Collector super-peer where the last is waiting for all peers to finish their phase 1 until it defines the phase2 bottom and broadcasts the suitable thresholds to them. A fully distributed run entails good synchronization of threads which comes from a simultaneous satisfaction of each peer's request. Our claim is based on the output files of each run. Since we write into files the execution of the algorithm, this action causes some extra delay to the execution time except from the sleep statement. Also we have to take into account the additional time needed to run each cluster of peers in each host computer.

By considering average execution times, we could observe the following: There is a linear increase in execution time which is sharper when the number of peers is above 30. The other observation is that we got better average times when we had zipfian distribution rather than normal. This is clearly depicted in Figure 4. However we could not make a safe conclusion since in each case of peers there are runs with smaller and larger values for both normal and zipfian scores. Also it is remarkable that the number of top-k candidate objects (final objects) is too small (10-28) when we have at least one patch phase. This is due to the pruning of top-k candidate set that occurs in each patch phase. In this way, we save in processing cost during the last phase, when we have to calculate the real scores for the final objects in order to return the top-k objects. In one case (normal distribution, 35 peers) when 23 patch phases run, the top-k candidate set was equal with the number of requested objects (10), so the algorithm directly returned the top-k objects.

Experiment 2

In the previous experiment all peers owned the same number of standard objects. In a more realistic scenario where each peer maintains its own data in its database, the number of objects is different. In experiment 2 each peer owns a number of objects and their corresponding scores according to the formula: *Total_Objects_Peer [i] = Standard_Objects + plus * i* where Standard_Objects = 30 and plus = 4. Thus, each peer owns at least 34 common objects (since i>=1) with different corresponding scores plus a number of other objects. By following this model the total objects that had been stored were 210 to 2760. For 5, 10, 20 and 30 peers the maximum discrete objects were correspondingly: 53, 70, 110 and 150. In this experiment we also included the case where no any sleep statement was set. Table 5 shows the corresponding results. Our intention was to compare same cases between experiments 1 and 2. Also in Figure 5 we can see graphically the average execution times of experiment 2, for the case where sleep statement was enabled in HT-p2p system.

Peers	Distribution	Seen Objects [Final Objects]	Patch Phases	Time (ms) [Average] Sleep Statement Enabled	Time (ms) [Average] Sleep Statement Disabled
5	normal	50 [10]	3	16610, 16904, 16952 **[16822]**	2141, 2375 4484 **[3000]**
5	zipfian	46 [10]	3	15593, 16906, 16625 **[16374]**	2140, 2141, 2562 **[2281]**
10	normal	69 [69]	0	16167, 16328, 17346 **[16613]**	5937, 6797, 8220 **[6984]**
10	zipfian	54 [54]	0	16832, 17390, 18140 **[17454]**	4953, 5453, 5609 **[5338]**
20	normal	110 [110]	0	31453, 41688, 49360 **[40833]**	
20	zipfian	102 [102]	0	26656, 59704, 64688 **[50349]**	
30	normal	150 [150]	0	48932, 52142, 96158 **[65744]**	
30	zipfian	137 [137]	0	47219, 59611, 78876 **[61902]**	

Table 5. Results of experiment 2

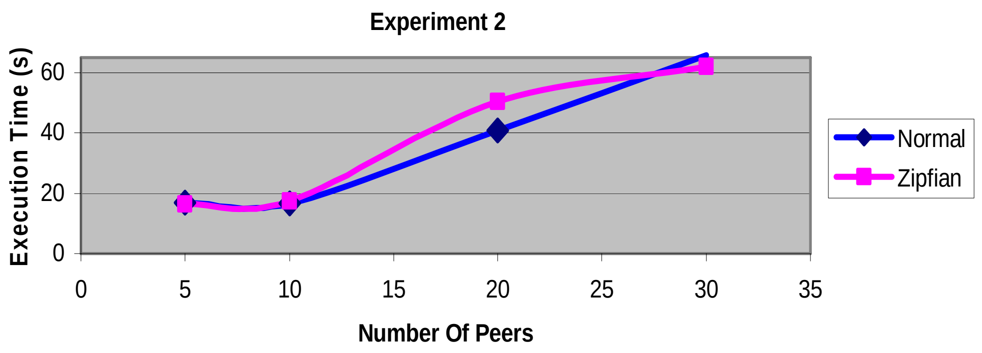

Fig. 5. Average time as the number of peers increases and sleep statement was enabled (exp.2)

Comparing the corresponding result tables, we conclude in general that execution times are better in experiment 2 rather than experiment 1. This is due to the fewer objects (and scores) that are transferred and processed during the execution of experiment 2. The differences in execution times are more noticeable when we had large number of contributor peers (e.g. 30 peers). However the fluctuations on time for the same cases also appeared in these experiments. Thus there is one case (zipfian, 20 peers) where this phenomenon was extreme, and so we got worst average execution time as compared with the corresponding case of experiment 1. Our possible explanation is again the bad synchronization of threads and as such the unsatisfied distributed running of HT-p2p's phases. Our findings about the delay of the sleep statement and the better performance (from object's aspect) of zipfian distribution against normal were confirmed in experiment 2 as well. With normal distribution, the HT-p2p algorithm comes close to the naïve algorithm (max number of objects) if we consider only the number of seen and final objects, but without taking into account object positions. For this reason we proceed to the next experiments.

Experiment 3

In experiment 3 we used the same random walk model as it was used in [2] in order to produce synthetic data sets. Specifically each object's score in peer i was given by the formula: *S[i] = S[i-1] + c * S[1]* where i>=1 and S[1] is the score of this specific object in peer_1. By varying c we can make cases where the object rankings are more or less similar in each peer. Our intention was to check how the algorithm behaves under situations where we had similar enough scores for the same object across the contributor peers. That means that the object's position in each sorted list of scores was similar. To archive this, we set c in small values. Each peer owned the same 150 objects according to the setting of experiment 1. In this experiment we made top-10 queries across 10, 20 and 30 contributor peers and we used three kinds of data distribution: zipfian, normal and uniform distribution. In peer_1 our random zipfian generator returned values from range [1, 499], the normal generator as it used the SSF net package [32] returned values too close to "150". Also through the use of [32] we used a random generator of uniform distribution which returned values from range [5, 499] to peer_1. All results are shown in Table 6.

Peers	**Distribution**	**C**	**Seen Objects [Final Objects]**	**Patch Phases**	**Average Time (ms)**
10	normal	**0.1**	**150 [150]**	5	17047
10	normal	**0.01**	**150 [150]**	5	16344
10	normal	**0.001**	**27 [27]**	5	18539
10	zipfian	**0.1**	**16 [16]**	5	14813
10	zipfian	**0.01**	**11 [11]**	5	19032
10	zipfian	**0.001**	**10 [10]**	5	17313
10	uniform	**0.1**	**41 [41]**	5	17766
10	uniform	**0.01**	**15 [14]**	5	17516
10	uniform	**0.001**	**11 [10]**	3	16436
20	normal	**0.1**	**150 [150]**	10	31078
20	normal	**0.001**	**62 [62]**	10	34025
20	zipfian	**0.1**	**150 [10]**	8	32188
20	zipfian	**0.001**	**150 [11]**	6	26188
20	uniform	**0.1**	**109 [83]**	10	23188
20	uniform	**0.001**	**88 [79]**	10	20782
30	normal	**0.1**	**150 [150]**	15	42454
30	normal	**0.001**	**107 [109]**	15	34475
30	zipfian	**0.1**	**150 [10]**	20	74220
30	zipfian	**0.001**	**150 [11]**	20	51501
30	uniform	**0.1**	**99 [94]**	13	56547
30	uniform	**0.001**	**67 [53]**	20	66735

Table 6. Results of experiment 3

As we observed for 10 peers when c was 0.1 for zipfian and uniform distribution the results were surely good since the number of seen and final objects were 27 and 41 for each distribution respectively. If we compare the corresponding case of zipfian in experiment 1 this value was 128, while the naïve's value is 150. Although for c=0.1 with normal distribution the algorithm again return the maximum number of final and seen objects. For this reason we set c to 0.01 and 0.001. The improvement in final and seen objects continued in both cases of zipfian and uniform distribution. In normal distribution for c = 0.001 we got 27 final and seen objects, a value surely good comparing with the value of the corresponding case in experiment 1 (150). Our conclusion is that the algorithm processed and transferred fewer (Object, Score) pairs when the c was 0.001 for all distributions This could be explained by the fact that top-k candidate objects in early phases are in practice the final top-k objects since their rankings are too similar. Hence, there is no reason for computing more partial sums for other objects in later phases (3 and 4) of HT-p2p algorithm.

Our considerations for these synthetic data sets were confirmed when we increased the number of contributor peers from 10 to 20 and 30 for normal and uniform distribution. In zipfian distribution we observed that although we got too small number of final objects, the algorithm examined 150 objects. Our intuition is that since in peer_1 a lot of objects had the value 1, as the number of peers was increased, more values were close enough and as such the algorithm had to examine the corresponding objects them during its execution. This probably didn't happen in uniform distribution, because our uniform scores took a lot of big values in peer_1 from which the synthetic data sets are produced. Also if we compare the corresponding cases for 10, 20 and 30 peers between experiment 1 and 3 we could see that in average for synthetic data sets we got better execution times. This could explain the saving of communication and processing cost of manipulated seen and final objects.

Experiment 4

In experiment 4 we measured the volume (in bytes) that is transferred in a full running instance of HT-p2p. These bytes are come from the transferred (Object, Score) pairs with/without the extra control information in each peer. We also compare the corresponding estimated transferred pairs for the case of Naïve Algorithm. We took 5 different cases from previous experiments: 10 peers with zipfian distribution, 10 peers with synthetic normal and zipfian distribution (C=0.001) and 30 peers with normal and zipfian distribution of scores. Table 7 depicts the results of experiment 4 and Figure 6 shows them graphically. The main conclusion of this experiment is that HT-p2p performs much better in all cases. Even in the last case where we had the maximum number of seen and final objects it transferred fewer bytes than the naïve algorithm.

Running Case	Seen Objects [Final Objects]	Patch Phases	NAÏVE Transferred Pairs (KB)	HT-p2p Transferred Pairs (KB)	HT-p2p Total Transferred Bytes (KB)
10 peers zipfian	128 [128]	0	10,3	**5,11**	**5,32**
10 peers zipfian Synthetic (C=0.001)	10 [10]	5	27, 4	**4,57**	**5,17**
10 peers normal Synthetic (C=0.001)	27 [27]	5	33,6	**6,14**	**6,54**
30 peers zipfian	144 [144]	0	31,1	**18,3**	**19,2**
30 peers normal	150 [150]	0	98.1	**92,4**	**93,7**

Table 7. Results of experiment 4 – Transferred bytes

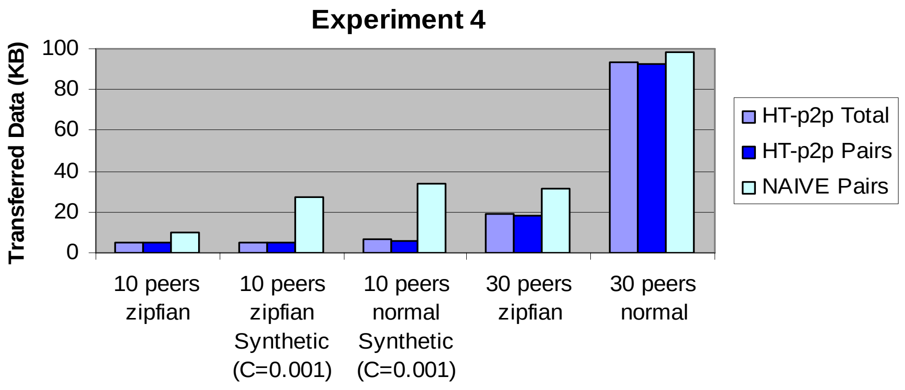

Fig. 6. Results of experiment 4 – Transferred bytes for 5 running cases

Also the superiority of HT-p2p is clearer when we have synthetic data sets for normal and zipfian distribution. At these cases although we had 5 patch phases, which entails more communication cost (transferred pairs), the total number of transferred bytes was too low comparing with the naïve case. This could be explained by the fact that top-k candidate objects in early phases are in practice the final top-k objects since their rankings are too similar. Hence, for the later phases less pairs are requested by HT-p2p and finally transferred across the p2p network. We should note that scores in normal distribution were double values with 14 digits accuracy while in zipfian the scores were integers cast to double. This explains the differences in transferred bytes comparing normal and zipfian scores for same cases. We also confirmed that the number of additional transferred bytes of control information is too low by comparing the total transferred bytes with the transferred pairs of HT-p2p.

Latest Experiments

Our last experiments included the validation that the HT-p2p algorithm like its ancestor (HT), does not get affected by the number of k. So we run 10 contributor peers and varied k from 5 to 140, while the discrete objects were 150 for both normal and zipfian distribution. We noticed that HT-p2p execution time and the processed objects do not seem to be affected by the value of k. As well, to validate the top-k results we setup 5 MySQL [33] databases (MySQL Server Version 5.0.24) and we tested HT-p2p algorithm under a total realistic p2p scenario. We calculated the top-k results and we validated them as true using appropriate queries.

At this point we should note that, the intention of our experiments was not to check the speed of HT-p2p algorithm - this could come from a simulation framework with large numbers of participant peers- but to test it under real conditions and with different parameters. Current plans also include the evaluation of HT-p2p+ and its comparison with the HT-p2p algorithm. We have already measured how the extra methods of HT-p2p+ affect the overall performance. In the case where no extra contribution from peers sending scores of unseen objects is needed for the calculation of real scores, we have noticed that the most time consuming operation is the detection of Collector super-peer which is achieved through a discovery message to the super-peer backbone.

Conclusions - Discussion

The results of our evaluation showed that our strategy could be adapted in a peer-to-peer network were peers host data in different data distributions under different scenarios. HT-p2p performs better when it has synthetic data with quite similar object rankings across databases of peers although it can work for all cases. As well, the linear increase in execution time of HT-p2p as the peers are grown up and the limited transferred bytes in standard communication rounds conduce to a promising top-k strategy.

In general efficient search in p2p networks could be improved by applying appropriate caching mechanisms in both query routing and processing strategy. In the first case a caching at schema-level (i.e. path expressions) at each cluster of Super-Peers for a specific query could give us the chance to apply the routing algorithm to a small subset of relevant peers. In the second case a caching at Originator/Collector Super-Peers for all processed top-k queries that had been answered in the past could give the opportunity to return the top-k results without running again an HT-p2p instance. In both cases we surely gain time and save communication cost. The appropriate cache replacement strategy should be determined in each case.

However since the general performance of the algorithm is affected by the execution of processing steps in a distributed way it would be meaningful to prevent the involuntary delay at phase2. Thus an extensibility suggestion could be the assessment of a specified timeout limit for each peer. Specifically, in the case when a peer could not overtake to finish its phase1 then its responsible super-peer could exclude it from the set of relevant participant peers at the specific running instance of HT-p2p. In addition it is desirable in the future to support of top-k join queries in order to have richer top-k query capabilities. The approach of [34] introduces some ideas that could be applied for p2p systems under some circumstances and with the suitable adaptations that need to be defined. Therefore, in general the problem of supporting top-k join queries is an open research topic for large scale distributed systems like peer-to-peer systems.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American, 284(5) pp. 34-43, (2001).
2. Yu, H., Li, H., Wu, P., Agrawal, D., Abbadi, A.E.: Efficient Processing of Distributed Top-k Queries. In Proceedings of the 16th International Conference on Database and Expert Systems Applications (DEXA 2005)
3. The JXTA Platform. http://www.jxta.org
4. Nejdl, W., Wolpers, M., Siberski, 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 World Wide Web Conference (WWW 2003)
5. Löser, A., Wolpers, M., Siberski, W., Nejdl, W.: Efficient Data Store Discovery in a Scientific P2P Network. In Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data, In Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)
6. Löser, A., Naumann, F., Siberski, W., Nejdl, W., Thaden, U.: Semantic Overlay Clusters within Super-Peer Networks. In Proceedings of the 1st International Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2003)

7. Tempich, C., Löser, A., Heinzmann, J.: Community Based Ranking in Peer-to-Peer Networks. In Proceedings of the 4th International Conference on Ontologies, Databases and Applications of Semantics (ODBASE 2005)
8. 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 World Wide Web Conference (WWW 2004)
9. Kokkinidis, G., Christophides, V.: Semantic Query Routing and Processing in P2P Database Systems: The ICS-FORTH SQPeer Middleware. In Proceedings of the 3rd Hellenic Data Management Symposium (HDMS 2004)
10. Fagin R. et. al.: Optimal Aggregation Algorithms for Middleware. In Proceedings of Symposium on Principles of Database Systems (PODS 2001)
11. 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 International Semantic Web Conference (ISWC 2004)
12. Theobald, M., Weikum, G., Schenkel R.: Top-k Query Evaluation with Probabilistic Guarantees. In Proceedings of 30th International Conference on Very Large Data Bases (VLDB 2004)
13. Michel, S., Triantafillou, P., Weikum, G.: Klee: A Framework for Distributed Top-k Query Algorithms. In Proceedings of 31st International Conference on Very Large Data Bases (VLDB 2005)
14. Hose, K., Karnstedt M., Sattler, K-U., Zinn D.: Processing Top-n Queries in P2P-Based Web Integration Systems with Probabilistic Guarantees. In Proceedings of 8th International Workshop on the Web & Databases (WebDB 2005)
15. Sartiani, C.: An Architecture for First-n and Top-n Queries in XML P2P Databases. Universita di Pisa (2005)
16. Schlosser, M., Sintek, M., Decker S., Nejdl, W.: HyperCuP- Hypercubes, Ontologies and Efficient Search on P2P Networks. In Proceedings of International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002)
17. Balke, W-T., Nejdl, W., Siberski W., Thaden, U.: Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. In Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)

18. Bruno, N., Gravano, L., Marian, A.: Evaluating Top-k Queries over Web-Accessible Databases. In Proceedings of the 18th International Conference on Data Engineering (ICDE 2002)
19. Marian, A., Bruno, N., Gravano L.: Evaluating Top-k Queries over Web-Accessible Databases. ACM Transactions Database Systems. (2004) 29(2) 319-362
20. Cao, P., Wang Z.: Efficient Top-k Query Calculation in Distributed Networks. In Proceedings of the 23rd Symposium on Principles of Distributed Computing (PODC 2004)
21. Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E.: Extensible Markup Language (XML) 1.0, W3C Recommendation, (2000). http://www.w3.org/TR/2000/REC-xml-20001006/
22. Lassila, O., Swick, R. Resource Description Framework Model and Syntax Specification, W3C Recommendation, (1999). http://www.w3.org/TR/REC-rdf-syntax/
23. Tzitzikas, Y., Meghini, C., Spyratos, N.: Taxonomy-Based Conceptual Modeling for Peer-to-Peer Networks. In Proceedings of the 22nd Conference on Conceptual Modeling (ER 2003)
24. Crespo, A., Garcia-Molina H.: Semantic Overlay Networks, for P2P Systems. Technical Report, Stanford University (2003)
25. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl M., RQL: A Declarative Query Language for RDF. In Proceedings of the 11th Int. World Wide Web Conference (WWW 2002)
26. Peter, H., Broekstra, J., Eberhart, A., Volz, R.: A Comparison of RDF Query Languages. In Proceedings of the 3rd International Semantic Web Conference (ISWC2004)
27. Comparison of RDF Query Languages. http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/
28. Dean, M., Schreiber, G., (Editors), Harmelen, F-V., Hendler, J., Horrocks I., McGuinness D., Patel-Schneider, P., Stein, L.: Web Ontology Language (OWL). http://www.w3.org/TR/owl-ref/
29. Peng, G.: CDN: Content Distribution Network, Stony Brook University, USA (2003)
30. http://java.sun.com/docs/books/tutorial/essential/threads/
31. The Apache ANT Project. http://ant.apache.org/
32. The SSFNet packages. http://www.ssfnet.org/javadoc/
33. The MYSQL database. http://www.mysql.com
34. Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting Top-k Join Queries in Relational Databases. VLDB Journal (2004) 13(3): 207-221