Efficient Early Top-k Query Processing in Overloaded P2P Systems

William Kokou Dedzoe, Philippe Lamarre, Reza Akbarinia, Patrick Valduriez

Efficient Early Top-k Query Processing in Overloaded P2P Systems

William Kokou Dédzoé[1], Philippe Lamarre[1], Reza Akbarinia[2], and Patrick Valduriez[2]

[1] LINA, University of Nantes, France
{William.Dedzoe, Philippe.Lamarre}@univ-nantes.fr
[2] INRIA and LIRMM, France
{Reza.Akbarinia, Patrick.Valduriez}@inria.fr

Abstract. Top-k query processing in P2P systems has focused on efficiently computing the top-k results while reducing network traffic and query response time. However, in overloaded P2P systems (with very high query loads), some peers may take a long time to answer, thus making the user wait a long time to obtain the final top-k result. In this paper, we address this problem, which we reformulate as early top-k query processing in P2P systems. First, to complement response time, we introduce two new metrics, stabilization time and cumulative quality gap, with which we formally define the problem. Then, we propose an efficient algorithm that dynamically adapts to query loads of peers in order to return to the user top-k results as soon as possible, without waiting for the final result. We validated our solution through simulations over a real dataset. The results show that our solution significantly outperforms baseline solutions by returning high quality top-k results to users in much better times.

1 INTRODUCTION

Top-k query processing in P2P systems has received a lot of attention [7, 18, 19, 1, 3]. The main reason for such interest is that they reduce the network traffic and avoid overwhelming the user with large numbers of uninteresting answers, which are resource-consuming. Despite the fact that these top-k query processing solutions reduce network traffic, finding an exact top-k in overload P2P (as the result of very high query loads) with some peers taking a long time to answer, may lead to long waiting times for users. By user waiting time, we mean the time that the user must wait to receive the final top-k result set for a top-k query.

In overload P2P systems, one of the main reasons for long waiting times is that usually with the existing approaches (e.g. [1] and [7]) there is no priority for the queries for which a peer can return high quality answers, i.e. queries are processed in order of their arrival. Therefore these solutions are not suitable for reducing the user waiting time in large scale applications which can be built on top of P2P systems, e.g. P2P web search engines. Such applications may easily become overloaded due to the high number of queries submitted to the system.

In this paper, we address the problem of reducing user waiting time of top-k query processing over horizontally partitioned data stored on peers in the

presence of high query loads in P2P systems. We reformulate this problem as *early top-k query processing in P2P systems*. To the best of our knowledge, this is the first work that deal with this problem in overload P2P systems. We revisit top-k query processing in P2P systems by considering two new metrics to complement response time: stabilization time and cumulative quality gap. Then, we propose an efficient algorithm that dynamically adapts to query loads of peers so as to return to the user top-k results as soon as possible, without waiting for the final results.

This paper makes the following contributions.

- We formally define the problem of early top-k query processing in P2P systems using both stabilization time and cumulative quality gap.
- We propose QUAT[3], an efficient algorithm for early top-k query processing. In QUAT, each peer maintains a semantic description of its local data and the semantic descriptions of its neighborhood (i.e. the semantic descriptions of data owned locally by its direct neighbors and data owned locally by these neighbors direct neighbors). These semantic descriptions allow peers to prioritize the queries that can provide high quality results, and to forward them in priority to the neighbors that can provide high quality answers.
- We validate our solution through a thorough experimental evaluation using a real-world dataset. The results show that QUAT significantly outperforms baseline algorithms by returning faster the final top-k results to users. They also demonstrate that in the presence of peer failures, QUAT provides top-k results with good accuracy compared to baseline algorithms.

The rest of this paper is organized as follows. In section 2, we make precise the P2P system model that we consider, with basic definitions regarding top-k queries. Section 3 defines the early top-k query processing problem. In Section 4, we present the QUAT algorithm. Section 5 presents how peers build and maintain routing indices based on their local and neighbors semantic descriptions for top-k query processing. In Section 6, we give our performance evaluation of QUAT. Section 7 discusses related work. In Section 8, we conclude.

2 P2P SYSTEM MODEL

In this section, we first present a general model of unstructured P2P systems which is needed for describing our solution. Notice that, our solution is not limited to unstructured P2P systems, but it can easily to be adapted to peers organized in a super-peer network. Next, we provide a model and definitions for top-k queries.

2.1 System Model

We model an unstructured P2P network of n peers as an undirected graph $G = (P, E)$, where $P = \{p_0, p_1, \cdots, p_{n-1}\}$ is the set of peers and E the set of

---

[3] Quality-based E<u>a</u>rly <u>T</u>op-k Query Processing refers to Khat, an African plant whose leaves are chewed as a stimulant.

connections between the peers. For $p_i, p_j \in P, (p_i, p_j) \in E$ denotes that p_i and p_j are neighbors. We denote by $N(p_i)$, the set of peers to which p_i is directly connected, so $N(p_i) = \{p_j | (p_i, p_j) \in E\}$. The value $\|N(p_i)\|$ is called the degree of p_i. The average degree of peers in G is called the *average degree* of G and is denoted by φ. Each peer $p \in P$ holds and maintains a set $D(p)$ of data items such as documents or relational data (i.e. tuples).

Let c_i be the number of queries which a peer p_i can process per time unit. We call c_i the capacity of p_i. If a peer receives queries from its neighbors at a rate higher than its capacity c_i, then the queries are queued until the receiving peer processes these queries. Note that the maximal number of connections (communication channels) which a peer can open simultaneously with its neighbors is proportional to the capacity of the peer. However, peers may set this number lower than the maximal value if they wish to.

In our model, the query is forwarded from the query originator to its neighbors until the Time-To-Live value of the query decreases to 0 or the current peer has no peer to forward the query. So the query processing flow can be represented as a tree, which is called the query forwarding tree.

2.2 Top-k Queries

We model each top-k query q by a tuple $< qid, \bar{q}, ttl, k, f, p_0 >$ such that qid is the query identifier, $\bar{q}$ is the query itself (e.g. SQL query), $ttl \in \mathbb{N}$ (Time-To-Live) is the maximum hop distance set by the user, $k \in \mathbb{N}^*$ is the number of results requested by the user, f is a scoring function that denotes the score of relevance (i.e. the quality) of a given data item to a given query and $p_0 \in P$ the originator of query q. We assume that the data scores are in $[0, 1]$. A top-k result set of a given query q is the k top results among data items owned by all peers that receive q. The data item in top-k result set having the lowest score is called the *mink* of that top-k result set.

2.3 Peer Semantic Description

In our system, each peer is described by a synthetic semantic description based on the data items owned by the peer. The approach of building this synthetic semantic description is out of the scope of this paper. We assume that it is obtained through a description aggregation function which takes as input a set of data items and generates a single description of these data items, e.g. as in [9] and [2]. We make the following assumptions regarding the description aggregation function:

(i) It is incremental, i.e. a peer that adds or removes a data item does not cause a total reconstruction of its semantic description.
(ii) It is composable, i.e. is possible to create a single semantic description using two or more semantic descriptions.
(iii) It is optimistic, i.e. the estimation of a top-k query's result quality with respect to a semantic description should not be lower than the exact scores of data (i.e. data which are used to build this semantic description).
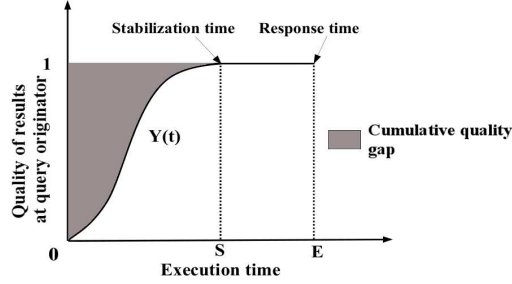
Fig. 1. Quality of top-*k* results at the query originator wrt. execution time

Notice that these properties of the semantic descriptions are realistic as it was demonstrated in [2].

3 Problem Definition

Let us first give our assumptions regarding schema management in the unstructured P2P system. We assume that peers are able to express queries over their own schema without relying on a centralized global schema. Several solutions have been proposed to support decentralized schema mapping and we simply assume it is provided using one of the existing techniques, *e.g.* [13]. In the following, we first give some definitions and formally state the problem.

3.1 Preliminaries

To process a top-*k* query in a P2P system, our approach provides intermediate results to users as soon as peers process the query locally. This allows users to progressively see the evolution of their query execution by receiving intermediate results. Notice that at some point of query execution, the top-*k* intermediate results received by the user may not change any more, because the user has already received all top-*k* results. We denote this point as the **stabilization time** (see Figure 1). The stabilization time may be much lower than the response time (when there is no more top-*k* result).

Our goal is to return high-quality results to the user as soon as possible. To capture this, we introduce the *quality evolution* concept as follows. Given a top-*k* query q, we define the quality evolution $Y(t)$ of q at time t as the sum of scores of q's intermediate top-*k* results at t and q's originator. To be independent of the scoring values (which can be different from one query to another), we normalize the quality evolution of a query. With this in mind, we divide the quality evolution of a given query by the sum of scores of the final top-*k* results of that query. Thus, the quality evolution values are in the interval $[0, 1]$ and the quality of the top-*k* final results is equal to 1.

The quality of intermediate top-*k* results at the query originator evolves during query execution. Let us now introduce the **cumulative quality gap**, which

is the sum of the quality difference between intermediate top-k result sets received until the stabilization time and the final top-k result set (see Figure 1). Notice that the smaller is the cumulative gap the higher is the quality of intermediate results returned to the user. We formally define the cumulative gap as follows.

Definition 1 *Cumulative quality gap*. *Let q be a top-k query, $Y(t)$ the quality evolution of q at time t at the query originator and s be the stabilization time of q. The cumulative quality gap of the query q, denoted by c_{qg} is:*

$$c_{qg} = \int_0^s (1 - Y(t))\, \mathrm{d}t = s - \int_0^s Y(t)\, \mathrm{d}t \tag{1}$$

In this paper, we address top-k query processing in overloaded P2P systems in which peers may receive many queries in a short time period. We define stabilization time and cumulative quality gap for time periods, and our objective is to develop algorithms that are efficient in terms of them.

Definition 2 *Stabilization time over a period*. *Given a time period $T = [t_1, t_2]$, let Q be the set of top-k queries issued during T. Then the stabilization time over T, denoted by S_T is:*

$$S_T = \frac{\sum_{q \in Q} s(q)}{\|Q\|} \tag{2}$$

where $s(q)$ is the stabilization of query $q \in Q$.

In other words, S_T is the average of the stabilization time for the queries issued in T.

Definition 3 *Cumulative quality gap over a period*. *Given a time period $T = [t_1, t_2]$, let Q be the set of top-k queries issued during T. Then the cumulative gap over T, denoted by C_{qgT} is:*

$$C_{qg[t_1,t_2]} = \frac{\sum_{q \in Q} c_{qg}(q)}{\|Q\|} \tag{3}$$

where $c_{qg}(q)$ is the cumulative gap of query $q \in Q$.

In other words, C_{qgT} is the average of the cumulative gap for the queries issued in T.

We do not use the proportion of the final top-k results in intermediate top-k results (i.e. precision) to characterize early top-k algorithm because this metric does not express the fact of returning the high quality results as soon as possible to users.

3.2 Problem Statement

Given a time period T, let S_T and C_{qgT} be the stabilization time and cumulative quality gap over T respectively. Our goal is to reduce S_T and C_{qgT} while providing the correct top-k result sets.

4 QUAT Top-k QUERY PROCESSING

In QUAT, each peer maintains a semantic description of its local data and the semantic descriptions of its neighborhood (i.e. the semantic descriptions of data owned locally by its direct neighbors and data owned locally by these neighbors direct neighbors). These semantics descriptions are used to create routing indices for top-k query processing. We give more details on the construction and maintenance of these routing indices in Section 5. Top-k query processing in QUAT proceeds in following phases: 1) query initialisation; 2) query forwarding; 3) local execution of the query by peers ; 4) bubbling up of the peers results for the query along the query forwarding tree.

4.1 Query initialisation

Query processing starts at the query originator, i.e. the peer at which a user issues a top-k query q. Note that the scoring function f and the number of results k wished by the user are specified in q. The query originator performs some initializations. First, it sets ttl which is either user-specified or default. Second, it creates a unique identifier qid for q which is useful to distinguish between new queries and those received before. qid is made of a unique peer identifier and a query counter managed by the query originator. Then, q is included in a message that is broadcast by the query originator to its reachable neighbors.

4.2 Query Forwarding

In a classical approach of query forwarding, when a peer p receives a query q, p forwards q in parallel to all its neighbors. The drawback of this approach is that is very resource consuming and may collapse the system in period of high query loads. To deal with this problem in our approach the maximum number of parallel connections m, a peer p can open for a query depends on the query load of p (i.e. the number of queries that are waiting to be forwarded by p). In our approach, we use the local query loads of each peer to set its value of m. However, it is possible to take into account loads of neighbors of peers and the overall load of the system if they can be obtained. In our approach, each peer p uses the semantic descriptions of its neighborhood to sort its neighbors based on the estimation of query's results quality which can be obtain from these neighbors. Given that the semantic description of a peer is optimistic ensure to peers to send queries in priority to neighbors which may provide high quality results. When all m connections are in use, no more connection can be established by p until one connection get released. Then p uses the free connection to communicate with

another neigbhor. The results returned by the peers which are already queried are used to query peers which are not yet queried. Moreover, these results are also used to avoid sending the query to peers who can not return results better than $mink$ of the current top-k result, i.e. by using the neighborhood semantic descriptions.

4.3 Local Query Execution

In a classical approach each peer executes locally incoming queries in the order of their arrival. However in period of high query load, executing locally queries in the order of their arrival may increase the waiting time for queries that a peer can provide better results for those it can provide results of low scores. To address this problem, in our approach the order in which incoming queries are executed locally by each peer p depends on the estimation of their results quality with respect to p's local semantic description. Given that the semantic description of a peer is optimistic ensure to peers to execute quickly queries which they may provide high quality results. The optimistic property of the semantic descriptions allows each peer p to avoid executing queries which its can not provide results which is lower than the $mink$ of the current top-k results at peer p. To summarize, in our solution, peers use intermediate results received from their children as well as the semantic descriptions to avoid executing locally some queries for which they do not have high quality results.

4.4 Bubbling Up Results

A naive solution to reduce the user waiting time is to return the top-k results from the peers directly to the query originator as soon as they have done executing the query. However, returning high numbers of results increases network traffic and can quickly cause a bottleneck at the query originator. For this in QUAT, when a peer submits a top-k query q, the local results of the peers that have received q are bubbled up to the query originator using query q's forwarding tree.

In QUAT, a peer's decision to send intermediate results is based on the improvement impact brought by its current top-k intermediate result set over the top-k intermediate result set it has already sent to its parent. This improvement impact can be computed by using the score of top-k results in the result set. Therefore, we introduce the notion of *score-based improvement impact*. Intuitively, the score-based improvement impact at a given peer for a given top-k query is the gain of score of peer's current top-k intermediate set compared to the top-k intermediate set it sent so far.

Definition 4 *Score-based improvement impact. Given a top-k query q, and peer $p \in \bar{P}$ (where $\bar{P}$ is the set of peers which received q), let T_{cur} be the current top-k intermediate set of q at p and T_{old} be the top-k intermediate set of q sent so far by p. The score-based improvement impact of q at peer p, denoted by $IScore(T_{cur}, T_{old})$ is computed as*

$$IScore(T_{cur}, T_{old}) = \frac{\sum\limits_{d \in T_{cur}} q.f(d, q.\bar{q}) - \sum\limits_{d' \in T_{old}} q.f(d', q.\bar{q})}{k} \qquad (4)$$

Notice that in Formula 4, we divide by k instead of $\|T_{cur} - T_{old}\|$ because we do not want that $IScore(T_{cur}, T_{old})$ be an average which would not be very sensitive to the values of scores. The score-based improvement impact values are in the interval $[0, 1]$.

To further reduce network traffic, QUAT does not bubble up the results (which could be large), but only their scores and addresses. A score-list is simply a list of k couples (ad, sc), such that ad is the address of the peer owning the data item and sc its score.

In QUAT, the minimum value that must reach the improvement impact before a peer sends newly received intermediate results to its parent is initially set by the application and is the same for all peers in the system. This threshold decreases as the query execution progresses. Using a dynamic threshold avoids the blocking problem of a static threshold when results having higher scores are bubbled up before those of lower score. Thus, we guarantee that low score results even tough they are in the final top-k results will not be returned at the end of the query execution.

To use a dynamic threshold approach, we need to compute the threshold value dynamically. We have identified two possible solutions for the dynamic threshold. The first one is to use an estimation of the query execution time. However, estimating the query execution time in large P2P system is very difficult because it depends on network dynamics, such as connectivity, density, medium access contention, etc., and the slowest queried peer. The second, more practical, solution is to use for each peer its local result set coverage to decrease the threshold. The local result set coverage of a peer for a given query is the proportion of peers in its sub-tree including itself which have already processed this query. We formalize this in Definition 5.

Definition 5 *Peer's local result set coverage*. *Given a top-k query, and $p \in \bar{P}$ (where $\bar{P}$ is the set of peers which received q), let $\mathcal{A}$ be the set of peers in the sub-tree whose root is p in the query q's forwarding tree. Let $\mathcal{E}$ be the set of peers in $\mathcal{A}$ which have already processed q locally. The local result set coverage of peer p for q, denoted by $Cov(\mathcal{E}, \mathcal{A})$, is computed using the following equation:*

$$Cov(\mathcal{E}, \mathcal{A}) = \frac{\|\mathcal{E}\|}{\|\mathcal{A}\|}$$

Peer's local result set coverage values are in the interval $[0, 1]$.

Computing the exact value of a peers local result set coverage incurs additional messages to the network, i.e. because each peer must send a message to its parent each time its local coverage result set value changes. To deal with this problem, we compute an estimation of this value instead of the exact value.

In our approach, the estimation is computed at the beginning by each peer based on the *ttl* received with the query and the average degree of peers in the

system. This value is updated progressively as the peers in its sub-tree bubble up their results. Indeed, each peer includes in each response message sent to its parent the number of peers in its sub-tree (including itself) which have already processed the query locally and the total number of peers in its sub-tree including itself. This couple of values is used in turn by its parent to estimate its local result set coverage. To decrease the improvement impact threshold used by a peer as the local result set coverage increases, we use a linear function that allows peers to set their improvement impact threshold for a given local result set coverage. Now let us define formally the threshold function.

Definition 6 *Threshold Function*. *Given a top-k query q and $p \in \bar{P}$ (where $\bar{P}$ is the set of peers which received q), the improvement impact threshold used by p during q's execution, is a monotonically decreasing function H such that:*

$$H : \begin{vmatrix} [0,1] \to [0,1] \\ x \quad \mapsto -\alpha * x + \alpha \end{vmatrix} \tag{5}$$

with $\alpha \in [0,1[$. Notice that x is a peer's result set coverage at given time and α the initial improvement impact threshold (i.e. $H(0) = \alpha$).

5 DISTRIBUTED SEMANTIC ROUTING INDICES

In this section, we first describe how to construct distributed semantic routing indices and then how to maintain them.

A semantic description routing index (or routing index for short) allows a peer to determine the priority of neighbors for sending the query when there is high query load in the system. It also allows a peer to avoid processing locally a query if its results for this query are not likely to bring anything to current top-k result set. Routing index is a data structure that, given a query, returns a list of neighbors, ranked according to their potential to answer the query. Let us now explain how these indices are created by peers. When a new peer p_i joins the system, it exchanges its own semantic description with those of its direct neighbors and these neighbors' direct neighbors (i.e peers which are 2 hops from p_i). Using these semantics descriptions, the peer p_i builds a semantic description table of its neighborhood. This table contains the identifier of each neighbor p_j of p_i and the aggregation of local semantic descriptions of p_j and p_j's direct neighbors. Semantic descriptions tables are used as routing indices for top-k query processing.

5.1 Maintaining Routing Indices

Updates of data owned by a peer may cause the modification of its semantic description. Therefore it is necessary that this modification be propagated to the neighbors to ensure accuracy of results returned to the user. A naive solution to maintain semantic descriptions up-to-date is to broadcast an update message containing the new semantic description of the peer and having $ttl = 1$ to all its

direct neighbors. Each neighbor which receives this update message, decreases the *ttl* of this message and sends it in turn to its neighbors (except to a peer from which it receives this message) until the *ttl* value reaches 0. The maintenance of a routing index after a modification in the peer's semantic description is done in $O(\varphi + \varphi^2)$ messages where φ is the average degree of peers in the system.

For efficiency reasons, we may choose not to send updates when the difference between the old and the new semantic description of a peer is not significant. By not sending minor updates, we can trade update cost for accuracy of the index.

Finally, a special update occurs in the case of churn of peers. When a peer p_i detects the disconnection of one of its neighbor p_j, p_i updates its routing index by removing the row for p_j. Then, it informs its direct neighbors by sending them an update message with $ttl = 1$. Each neighbor which receives this update message, decreases *ttl* by one and sends it in turn to its neighbors (until *ttl* reaches 0).

6 Performance Evaluation

In this section, we evaluate the performance of QUAT through simulation using PeerSim [11], an open source, Java based, P2P simulation framework. First, we describe our simulation setup, the metrics used for performance evaluation, the baseline top-k query processing approaches and the datasets used for experiments. Then, we study the effect of the query arrival rate on the performance of QUAT, and show how it scales up. Next, we investigate the effect of peers failures on the correctness of QUAT.

6.1 Setup

We implemented our simulation using the PeerSim simulator. We conducted our experiments on a machine with a 2.4 GHz Intel Pentium 4 processor and 2GB memory. The simulation parameters are shown in Table 1. We use parameters values which are typical of P2P systems [15]. The latency between any two peers is a normally distributed random number with a mean of 200 ms. Since users are usually interested in a small number of top results, we set $k = 20$ as default value. In our experiments we vary the network size from 1000 to 10000 peers . In order to simulate high heterogeneity, we set peers' capacities in our experiments, in accordance to [15] which measures peer capacities in the P2P system. Based on the results of [15], we generate around 10% of low-capable, 60% of medium-capable, and 30% of high-capable peers. The highly-capable peers are 3 times more capable than medium-capable peers and still 7 times more capable than low-capable ones. Each experiment is run for 2 hours, which are mapped to simulation time units. In all our experiments, we use $H(x) = -0.2x + 0.2$ as threshold function and the maximum number of connections that a peer p opens simultaneously for a query is set to $\|N(p)\| \div 2$, where $\|N(p)\|$ is degree of p.

6.2 Dataset

We conducted our experiments using HTTP server logs dataset. The Internet Traffic Archive [4] provides a huge HTTP server log with about 1.3 bil-

---
[4] http://ita.ee.lbl.gov

Table 1. Simulation parameters.

Parameters	Values
Latency	Normally distributed random number, Mean=200 ms, Variance=100
Number of peers	10000 peers
Average degree	4
ttl	9
k	20
Query arrival rate	50 queries per seconds

lion HTTP requests from the 1998 FIFA soccer world championship. We aggregated the information from this log into a relational table with the schema $Log(interval, userid, bytes)$, aggregating the traffic (in bytes) for each user within one-day intervals. This dataset is horizontally partitioned evenly among peers of the P2P system. Queries ask for the top-k active users, i.e. the k users with the highest traffic at given interval (like "June 1").

6.3   Metrics

In our experiments, to evaluate the performance of QUAT and that of baseline approaches, we use the following metrics:

(i) **Cumulative quality gap over a period:** see Section 3 for the definition.
(ii) **Stabilization time over a period:** see Section 3 for the definition.
(iii) **Response time over a period:** We report on the average response time of all queries submitted in the system over a given period. The response time is the time the query initiator has to wait until the top-k query execution is finished.
(vi) **Communication cost:** We measure the communication cost in terms of number of answer messages and volume of data which must be transferred over the network in order to execute a top-k query.
(vii) **Accuracy of results:** We define the accuracy of results as follows. Given a top-k query q, let V be the set of the k top results owned by the peers that received q, V' be the set of top-k results which are returned to the user as the response of the query q. We denote the accuracy of results by ac_q and define it as:

$$ac_q = \frac{\|V \cap V'\|}{\|V\|}$$

6.4   Baseline Approaches

In unstructured P2P systems, Fully Distributed (FD) [1] and As Soon As Possible (ASAP) [7] are baseline approaches for top-k processing over horizontally partitioned data stored on peers. In FD, each peer that receives the query executes it locally (i.e. selects the k top scores), and waits for its children's results.

After receiving all its children score-lists, the peer merges its k local top data items with those received from its children and selects the k top scores and sends the result to its parent. Unlike FD, in ASAP, a peer does not wait for all its children results before bubbling up results to its parent. Each peer (except the query originator) returns to its parent its intermediate results that have better qualities and thus may be in the final top-k.

6.5 Performance Results

In this section, we report the results of our experimentation. Due to space limitations, we only present the main results.

Effect of arrival query rate We study the effect of the query arrival rate on the performance of QUAT. For this, we ran experiments to study cumulative quality gap, stabilization time, response time and volume of transferred data while increasing the query arrival rate in the system from 50 to 300. Note that the other simulation parameters are set as in Table 1.
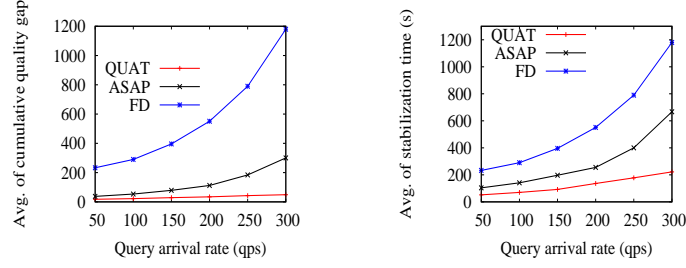
Figures 2(a) and 2(b) show respectively how cumulative quality gap and stabilization time over a period of 2 hours increase with the query arrival rate. The results show that the cumulative quality gap of QUAT is always much smaller than that of ASAP and FD, which means that QUAT returns much faster high quality results than ASAP and FD. The results also show that the stabilization time of QUAT is always much smaller than that of ASAP and FD. The reason is that in QUAT, peers prioritize the execution of queries that can produce high quality results. Figure 2(c) show that the response time of QUAT over a period of 2 hours is always much better than that of ASAP and FD. The main reason is that in QUAT, peers do not execute incoming queries for which they do not have interesting data, which helps peers to save their resources.

Figure 2(d) shows the volume of the increase of transferred data vs. query arrival rate. The results show that the volume of transferred data of QUAT is always lower than that of ASAP. The results also show that the difference between QUAT and FD's volume of transferred is not significant.
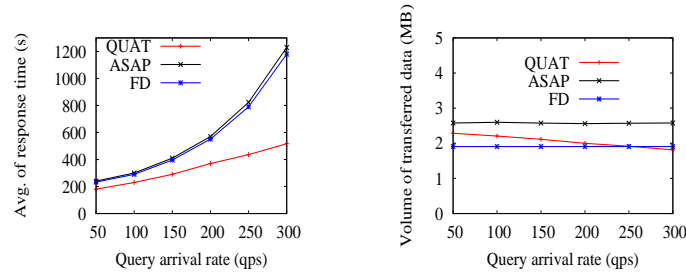
Effect of peers failures In this section, we investigate the effect of peers failures on the accuracy of top-k results. In our tests, we vary the value of fail rate and investigate its effect on the accuracy of top-k results. Figure 3 shows the accuracy for QUAT, ASAP and FD while increasing the fail rate, with the other parameters set as in Table 1. Peers' failures have less impact on QUAT than ASAP and FD. The reason is that QUAT returns high-score results to the user very quickly. However, when increasing the fail rate in ASAP and FD, the accuracy of top-k results decreases significantly because some score-lists are lost. Indeed, in FD, each peer waits for results of its children so in the case of a peer failure, all the score-lists received so far by that peer are lost.

7 RELATED WORK

Efficient processing of top-k queries is both an important and hard problem that is still receiving much attention [17, 14]. Several works have dealt with

(a) Cumulative quality gap vs. Query rate

(b) Stabilization time vs. Query rate

(c) Response time vs. Query rate

(d) Volume of transferred data vs. Query rate

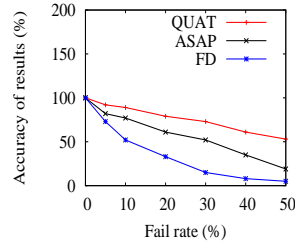**Fig. 2.** Impact of query loads on QUAT performance

Fig. 3. Accuracy of results vs. fail rate

top-k query processing in centralized database management systems [16, 10]. In distributed systems [5], previous work on top-k processing has focused on vertically distributed data over multiple sources, where each source provides a ranking over some attributes. The majority of the proposed approaches try to improve some limitations of the Threshold Algorithm (TA) [8]. Following the same concept, there exist some previous work for top-k queries in P2P over vertically distributed data. In [4], the authors propose algorithm called "Three-

Phase Uniform Threshold" (TPUT) which aims at reducing communication cost by pruning away intelligible data items and restricting the number of round-trip messages between the query originator and other nodes. Later, TPUT was improved by KLEE [12]. KLEE uses the concept of bloom filters to reduce the data communicated over the network upon processing top-k queries. It brings significant performance benefits with small penalties in result precision. However, theses approaches assume that data is vertically distributed over the nodes whereas we deal with horizontal data distribution.

For horizontally distributed data, there has been little work on P2P top-k processing. In [1], the authors present FD, a fully distributed approach for top-k query processing in unstructured P2P systems. Recently, FD was improved by ASAP [7]. We have briefly introduced FD and ASAP in section 6.4.

In [3], the authors present an index routing based a top-k processing technique for super-peer networks organized in an HyperCuP topology which tries to minimize the number of transfer data. The authors use queries statistics to maintain the indices built on super-peers. However, the performance of this technique depends on the query distribution.

Zhao *et al.* [19] use a result caching technique to prune network paths and answer queries without contacting all peers. The performance of this technique depends on the query distribution. They assume acyclic networks, which is restrictive for unstructured P2P systems.

There have been many works to deal with the problem of query load balancing by trying to distribute the load fairly over the peers of the system, e.g. [6]. However, in the current paper, our objective is not to balance the load, but to take it into account for reducing the user waiting time.

8 CONCLUSION

In this paper, we addressed the problem of top-k query processing in overloaded P2P systems. The objective is to reduce the user waiting time by returning high quality intermediate results as soon as possible, while avoiding high network traffic. For this, we revisited the problem of top-k query processing by considering two new metrics to complement response time: stabilization time and cumulative quality gap. Then, we proposed QUAT, an efficient algorithm that dynamically adapts to peer query loads in order to return to the user top-k results as soon as possible. QUAT allows users to progressively see the evolution of their query execution by receiving high quality intermediate results. We validated QUAT through extensive experimentation. The results show that QUAT significantly outperforms baseline algorithms by providing quickly high quality to users and by returning final top-k result to users in much better times. Finally, the results demonstrate that in the presence of peers' failures unlike baseline algorithms, QUAT provides top-k results with good accuracy.

References

1. R. Akbarinia, E. Pacitti, and P. Valduriez. Reducing network traffic in unstructured p2p systems using top-k queries. *Distributed and Parallel Databases*, 19(2-

3):67–86, 2006.

2. V. Anthony, P. Lamarre, C. Sylvie, and V. Patrick. Représentation optimiste de contenus dans les systèmes p2p. In *Journées Fancophones de Bases de Données Avancées (BDA)*, pages 1–18, 2009.

3. W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top k retrieval in peer-to-peer networks. In *Proceedings of Int. Conf. on Data Engineering (ICDE)*, pages 174–185, 2005.

4. P. Cao and Z. Wan. Efficient top-k query calculation in distributed networks. In *Proceedings of Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 206–215, 2004.

5. S. Chaudhuri, L. Gravano, and A. Marian. Optimizing top-k selection queries over multimedia repositories. *IEEE Transactions on Knowledge Data Engineering (TKDE)*, 16(8):992–1009, 2004.

6. A. Datta. Load balancing in peer-to-peer overlay networks. In *Encyclopedia of Database Systems*, pages 1627–1632. Springer US, 2009.

7. W. K. Dedzoe, P. Lamarre, R. Akbarinia, and P. Valduriez. Asap top-k query processing in unstructured p2p systems. In *Proceedings of IEEE Int. Conf on Peer-to-Peer Computing (P2P)*, pages 187–196, 2010.

8. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of Symposium on Principles of Database Systems (PODS)*, pages 102–113, 2001.

9. R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib. Summary management in p2p systems. In *Proceedings of Int. Conf on Extending Database Technology (EDBT)*, pages 16–25, 2008.

10. V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: a system for the efficient execution of multi-parametric ranked queries. In *Proceedings of ACM. Int Conf. on Management of Data (SIGMOD)*, pages 259–270, 2001.

11. M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

12. S. Michel, P. Triantafillou, and G. Weikum. Klee: A framework for distributed top-k query algorithms. In *Proceedings of Int. Conf. on Very Large Data Bases (VLDB)*, pages 637–648, 2005.

13. B. C. Ooi, Y. Shu, and K.-L. Tan. Relational data sharing in peer-based data management systems. *SIGMOD Record*, 32(3):59–64, 2003.

14. A. Radwan, L. Popa, I. R. Stanoi, and A. A. Younis. Top-k generation of integrated schemas based on directed and weighted correspondences. In *Proceedings of ACM Int. Conf on Management of data (SIGMOD)*, pages 641–654, 2009.

15. S. Saroiu, P. K. Gummadi, P. K. Gummadi, S. D. Gribble, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the Multimedia Computing and Networking Conference (MMCN)*, 2002.

16. M. Shmueli-Scheuer, C. Li, Y. Mass, H. Roitman, R. Schenkel, and G. Weikum. Best-effort top-k query processing under budgetary constraints. In *Proceedings of Int. Conf on Data Engineering (ICDE)*, pages 928–939, 2009.

17. A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *Proceedings of Int. Conf on Data Engineering (ICDE)*, pages 365–376, 2010.

18. A. Vlachou, C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. On efficient top-k query processing in highly distributed environments. In *Proceedings of ACM. Int Conf. on Management of Data (SIGMOD)*, pages 753–764, 2008.

19. K. Zhao, Y. Tao, and S. Zhou. Efficient top-k processing in large-scaled distributed environments. *Data and Knowledge Engineering*, 63(2):315–335, 2007.