SKYPEER: Efficient Subspace Skyline Computation over Distributed Data

Akrivi Vlachou[1]*, Christos Doulkeridis[1]†, Yannis Kotidis[1]
[1]Department of Informatics
Athens University of Economics and Business
Patision 76, 10434 Athens, Greece
{avlachou, cdoulk, kotidis}@aueb.gr

Michalis Vazirgiannis[1,2]‡
[2]GEMO Team
INRIA/FUTURS, France
mvazirg@aueb.gr

Abstract

Skyline query processing has received considerable attention in the recent past. Mainly, the skyline query is used to find a set of non dominated data points in a multi-dimensional dataset. While most previous work has assumed a centralized setting, in this paper we address the efficient computation of subspace skyline queries in large-scale peer-to-peer (P2P) networks, where the dataset is horizontally distributed across the peers. Relying on a super-peer architecture we propose a threshold based algorithm, called SKYPEER, which forwards the skyline query requests among peers, in such a way that the amount of transferred data is significantly reduced. For efficient subspace skyline processing, we extend the notion of domination by defining the extended skyline set, which contains all data elements that are necessary to answer a skyline query in any arbitrary subspace. We prove that our algorithm provides the exact answers and we present optimization techniques to reduce communication cost and execution time. Finally, we provide an extensive experimental evaluation showing that SKYPEER performs efficiently and provides a viable solution when a large degree of distribution is required.

1. Introduction

Skyline queries help users make intelligent decisions over complex data, where different and often conflicting criteria are considered. Such queries return a set of interesting data points that are not dominated by any other point on all dimensions [4]. Skyline queries are beneficial to distributed applications like web information systems [1]. However, as the number of participating sources increases, traditional client-server solutions are prone to bottleneck risks and therefore cannot scale. Furthermore, the amount of data increases at tremendous rates, making its central assembly at one location infeasible. Last, but not least, independent information sources usually do not wish to allow uncontrolled access to their local data repositories, mainly for privacy reasons.

Consider a global-scale web-based hotel reservation system, consisting of a large set of independent servers geographically dispersed around the world. Servers accept subscriptions by travel agencies, in order to advertise their hotels. Towards this goal, servers are willing to share some information about the data stored locally, without handing over the whole dataset. Such a system could potentially provide booking services over the universal hotel database, without requiring from each travel agency to register with each server. This need becomes even more important, due to the fact that the number of providers (and therefore data) increases at tremendous rates. The challenge is to enable users to pose interesting queries, such as skylines, over this network of servers, and retrieve those results that match a possibly different (each time) set of user-defined criteria. In general, given a set of d-dimensional objects, a *subspace skyline query* only refers on a subset of attributes of size $k \leq d$.

A promising solution for the design and deployment of global-scale applications such as the above, is the exploitation of peer-to-peer (P2P) technology. The P2P paradigm emerges as a powerful model for organizing and searching large data repositories distributed over independent sources [7]. In this work we explore how subspace skyline queries can be computed efficiently over a distributed and large-scale web information system modeled as a P2P network. Our approach relies on a super-peer architecture, where each peer holds only a portion of the entire dataset. The main contributions of our work are:

- We explore the implications of processing subspace skyline queries in large scale P2P networks, since ex-

*Partially funded by the PENED 2003 Programme of the EU and the Greek General Secretariat for Research and Technology.

†Partially funded by the PENED 2003 Programme of the EU and the Greek General Secretariat for Research and Technology.

‡Supported by the Marie Curie Intra-European Fellowship.

isting methods that have been developed for centralized systems can not be adapted to work efficiently in a P2P environment.

- We extend the notion of domination by defining the *extended skyline set*, which can provably answer arbitrary subspace queries in an efficient manner. We then present our algorithm, SKYPEER, which orchestrates the processing of a subspace skyline query using extended skyline sets.
- SKYPEER utilizes a thresholding scheme, in order to facilitate pruning of dominated data across the peers. We explore different strategies for (i) threshold propagation and (ii) result merging through the P2P network.
- We provide an extensive experimental evaluation of our algorithms and compare it against a straightforward skyline computation algorithm adapted to work in P2P networks.

The rest of this paper is organized as follows: Section 2 provides an overview of related work. In Section 3 we describe the P2P architecture employed and we present a baseline algorithm for subspace skyline queries. In Section 4 we introduce the extended skyline, which provides the foundation for our algorithm presented in Section 5. Our experimental results are presented in Section 6. Finally, in Section 7 we conclude the paper.

2. Related Work

Skyline computation has recently received considerable attention in the database research community. Two algorithms, BNL and DC are proposed in [4], while SFS [5], is based on the same principle as BNL, but improves performance by first sorting the data according to a monotone function. Tan *et al.* [16] propose the first progressive techniques, namely Bitmap and Index method. In [11], an algorithm based on nearest neighbor search on the indexed dataset is presented. Then, Papadias *et al.* [14] propose a branch and bound algorithm to progressively output skyline points on a dataset indexed by an R-Tree, with guaranteed minimum I/O cost.

Recently papers focus on algorithms to support subspace skyline retrieval. In [17] SUBSKY, a non-incremental algorithm, is presented, which transforms the multi-dimensional data to one-dimensional values, and indexes the dataset with a B-Tree. Pei *et al.* [15] and Yuan *et al.* [20] independently propose the SKYCUBE, which consists of the skylines in all possible subspaces. In [6] the problem of supporting constrained subspace skylines, which form the generalization of all meaningful skyline queries over a given dataset, was posed.

In spite of the importance of the skyline there is limited work on skyline calculation in decentralized and distributed environments. In [1] an algorithm for distributed environments was proposed. Unfortunately, their assumptions are hardly applicable to large-scale P2P systems. Furthermore, while they assume a vertical partitioning of the dataset across the Web sources, data in P2P systems is, typically, horizontally partitioned across the peers.

The first work that focuses on P2P skyline computation is presented in [9, 8]. The authors focus on Peer Data Management Systems (PDMS), where each peer provides its own data with its own schema. Their techniques provide probabilistic guarantees for the result's correctness. In comparison, our algorithms provably return *exact* answers to arbitrary subspace skyline computations. In [10], Huang *et al.* assume a setting with mobile devices communicating via an ad hoc network, and study skyline queries that involve spatial constraints. The authors present techniques that aim to reduce the costs of communication and reduce the execution time on each single device. Wu *et al.* [18] first address the problem of parallelizing progressive skyline queries on a share-nothing architecture. Their techniques enforce the skyline partial order, so that the system pipelines participating machines during query execution and minimizes inter-machine communication.

Super-peer networks [19], like KaZaa and eMule, have emerged as a viable solution for developing existing P2P systems, harnessing the merits of both centralized and decentralized systems. Edutella [13] is such a super-peer approach, based on HyperCup topology, where super-peers maintain indices of peer contents and routing indices, and searching is practically achieved through routing at super-peer level. Distributed top-k retrieval [12] and P2P ranking [2] have also attracted attention lately, motivating and relating to our research in P2P skyline query processing.

3. Skyline Computation in P2P Networks

Skyline query processing in P2P networks poses inherent challenges and demands non-traditional techniques due to the distribution of content and the lack of global knowledge. Relying on a super-peer architecture we propose a threshold based algorithm, called SKYPEER and its variants, for efficient computation of skyline points in arbitrary subspaces, reducing both computational time and volume of transmitted data.

3.1. Preliminaries and Definitions

We assume an unstructured P2P network of N_p peers. Some peers have special roles, due to their enhanced features, such as availability, storage capability and bandwidth capacity. These peers are called super-peers SP_i

Symbols	Description
d	Data dimensionality
k	Query dimensionality
n	Cardinality of the dataset
S	Dataset
S_i	Partition of dataset ($i = 1..N_p$)
N_p	Number of peers
N_{sp}	Number of super-peers ($N_{sp} << N_p$)
DEG_p	Degree of simple peer
DEG_{sp}	Degree of super-peer ($DEG_{sp} < DEG_p$)
SKY_U	Skyline of subspace U
SKY_{U_i}	Skyline of subspace U on super-peer SP_i
$ext\text{-}SKY_U$	Extended skyline of U
$ext\text{-}SKY_{U_i}$	Extended skyline of U on super-peer SP_i
t	Threshold

Table 1. Overview of symbols.

($i = 1..N_{sp}$), and they constitute only a small fraction of the peers in the network, i.e. $N_{sp} << N_p$. Peers that join the network directly connect to one of the super-peers. Each super-peer maintains links to simple peers, based on the value of its degree parameter DEG_p. In addition, a super-peer is connected to a limited set of at most DEG_{sp} other super-peers ($DEG_{sp} < DEG_p$).

In super-peer architectures, queries are typically routed first in the super-peer backbone and afterwards, if necessary, they are distributed to the peers that are connected to the super-peers. In general, super-peers maintain information about the peers they have been assigned, so that at query time, they can process a query without having to contact all peers. Note that one important performance parameter is the super-peer topology which influences the routing performance. In this work we assume that the super-peer topology is pre-defined and we focus on the optimization of interactions among super-peers and peers.

Each peer P_i holds n_i d-dimensional points, denoted as a set S_i ($i = 1..N_p$). Obviously the size of the complete set of points is $n = \sum_{i=1}^{N_p} n_i$ and the dataset S is the union of all peers' datasets S_i: $S = \cup S_i$. Given a space D defined by a set of d dimensions $\{d_1, d_2, .., d_d\}$ and a dataset S on D, a point $p \in S$ can be represented as $p = \{p[1], p[2], ..., p[d]\}$ where $p[i]$, is a value on dimension d_i. Each non-empty subset U of D ($U \subseteq D$) is referred to as a *subspace*. Without loss of generality, we assume that skylines are computed with respect to min conditions on all dimensions and that all values are non-negative. A point $p \in S$ is said to *dominate* another point $q \in S$ on subspace U, if on each dimension $d_i \in U$, $p[i] \leq q[i]$; and on at least one dimension $d_j \in U$, $p[j] < q[j]$. The *skyline of a subspace* $U \subseteq D$ is a set of points $SKY_U \subseteq S$ which are not dominated by any other point on subspace U. The points in SKY_U are called *skyline points* on subspace U. For a complete reference to the symbols used in this paper see Table 1.

3.2. A Baseline Algorithm

In the rest of this paper, we denote the querying peer P_{init}, henceforth referred to as *initiator* of a query[1]. In a distributed setting, a simple solution is that all peers send their local datasets to P_{init} where a centralized skyline algorithm is executed. However, this approach is infeasible in a large-scale P2P system due to the high incurred costs and is not considered further as candidate solution. Therefore, our premise is to locally evaluate as many parts of the query as possible. However, accurate skyline computation over widely distributed data, demands that all data is taken into account, since even a single point neglected could be part of the skyline and, thus, prune out other points already processed. Our main observation is that a point $p \in SKY_U$, only if there exists a partition S_i where p is a local skyline in U. Thus, each super-peer needs to collect from the associated peers only the skyline points of all subspaces. In the next section, we define an appropriate set, called *extended skyline*, with low computation cost.

Given the locally stored extended skyline, each super peer individually processes a subspace skyline request and transmits the results to the query initiator. Notice that some local skyline points may not belong to the global SKY_U. Thus, the query initiator needs to collect the results from all super-peers and merge them by discarding dominated points. This approach is considered as baseline, and will be henceforth referred to as "naive". In what follows we aim to improve the performance of the naive approach by reducing communication costs and processing time.

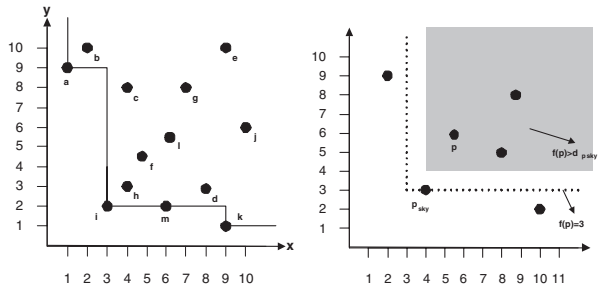
4. Extended Skyline

In order to avoid the transmission of all data in the P2P network, we need to calculate a subset of the original dataset that contains all the skyline points for any subspace. As shown in the following two Observations [20], the set of the skyline points of the global data space is not sufficient.

Observation 1. Given a set S of data points on dimension set D, for two subsets U and V of D (U,$V \subseteq D$), where $U \subseteq V$, there is no containment relationship between SKY_U and SKY_V.

Observation 2. Assume a set S of data points on dimension set D and two subsets U and V of D (U,$V \subseteq D$) such that $U \subset V$. Each skyline point q in SKY_U on dimension set V is either dominated by another skyline point p in SKY_U; or a skyline point in SKY_V.

[1]Notice that even though the initiator can be a simple peer, we use P_{init} to refer to the super-peer responsible for the simple peer.



(a) Dataset (b) Mapping

Figure 1. Skyline examples.

Based on the above observations, a skyline point q in SKY_U is either a skyline point in SKY_V or there is another data point p such that $p[a_i] = q[a_i]$ $(\forall a_i \in U)$ that dominates q on the dimension set $V - U$. Thus, a super-set of the union of all subspace skylines is the set of the global skyline points enriched with all points p for which $\exists i \exists q \in SKY_D$ $q[a_i] = p[a_i]$. Consider for example Figure 1(a) where e and k have the same x-value but k is a skyline point in contrast to point e which is not a skyline point in any subspace.

To reduce the computation overhead we compute a subset of the mentioned super-set that is able to answer any subspace skyline query. Therefore, we adjust the dominance definition to compute all necessary values coinstantaneously during the skyline calculation. We define the *extended-skyline (ext-skyline)* based on the *extended domination (ext-domination) definition*.

Definition 1. For any dimension set U, where $U \subseteq D$, p ext-dominates q if on each dimension $d_i \in U$, $p[i] < q[i]$. The ext-skyline ($ext\text{-}SKY_U$) is set of all points that are not ext-dominated by any other.

In the following, we prove that $ext\text{-}SKY_D$ is sufficient to answer any subspace skyline query correctly.

Observation 3. Every point that belongs to the skyline of U belongs also to the ext-skyline of U, i.e. $SKY_U \subseteq ext\text{-}SKY_U$.

Proof: Let $p \in SKY_U$ and $p \notin ext\text{-}SKY_U$. It follows that there is a point q that ext-dominates the point p in U. Based on the definition of the ext-skyline $\forall a_i \in U : q(a_i) < p(a_i)$. Therefore, based on the skyline definition we conclude that $p \notin SKY_U$, which leads to a contradiction.

Observation 4. Every point that belongs to the skyline of a subspace $V \subseteq U$ belongs to the ext-skyline of U, i.e. $SKY_V \subseteq ext\text{-}SKY_U, V \subseteq U$.

Proof: Based on Observation 2 we distinguish two cases. If p is a skyline point in U then Observation 3 guarantees that $p \in ext\text{-}SKY_U$. If p is not a skyline point in SKY_U based on Observation 2 there is a skyline point q in U such that $p(a_i) = q(a_i)$, $\forall a_i \in U$. Based on Observation 3 and the definition of ext-skyline we conclude that $q \in ext\text{-}SKY_U$ and $p \in ext\text{-}SKY_U$.

For example, in Figure 1(a), point m belongs to the ext-skyline, which is not the case with point e, since e is globally dominated by i, which in turn does not have any value equal to the attribute values of e. Notice, that neither e nor m belong to any subspace skyline.

5. The SKYPEER Algorithm

In this section we describe our algorithm for subspace skyline computation in a P2P environment. First, in a preprocessing phase (more on this in Section 5.3) each peer P_i computes the local ext-skyline of its dataset S_i and sends it to the associated super-peer. The super-peer calculates its own ext-skyline, denoted as $\cup ext\text{-}SKY_{D_i}$ $(i = 0..DEG_p)$, by merging the local skyline results. Based on Observation 4 the local ext-skyline is sufficient for a super-peer to determine if any of its peers P_i can contribute to the results of any skyline query on an arbitrary subspace $U \subseteq D$. In order to speed up the subspace skyline computation the multi-dimensional data is transformed into one dimensional values. This mapping is the topic of the next subsection.

5.1. Mapping

Recall that each super-peer maintains some portion of the dataset consisting of d-dimensional points. A query refers to a non-empty subset U of D. The values of all dimensions are assumed to be non-negative.

Inspired by [17], each d-dimensional point p is transformed to a one dimensional value $f(p)$ based on the formula:

$$f(p) = \min_{i=1}^{d}(p[i]) \quad (1)$$

Let $dist_U(p)$ denote the L_∞-distance of point p from the origin based on the dimension set U, i.e. $dist_U(p) = \max_{i \in U}(p[i])$.

Observation 5. Let p_{sky} be a skyline point in a subspace U. A point p for which the following inequality holds cannot be a skyline point in subspace U.

$$f(p) > dist_U(p_{sky}) \quad (2)$$

In Figure 1(b) we consider an example for the case of a two dimensional skyline query. Let p_{sky} be a skyline point. The $f(p_{sky})$ value equals to 3 showing that p_{sky} lies on the dashed line, while Observation 5 allows us to prune the shadowed area. Notice that the main difference of the proposed transformation to the one in [17] is that distances are calculated based on the origin of axes, while in [17] the maximum corner[2] is used. This method is not directly applicable to a distributed environment since the maximum corner is not known to all super-peers a priori. Recently, in [3] the idea of limiting the amount of data to be read by exploiting the value of a monotone function was studied.

[2]The right upper corner of the universe.

Algorithm 1 Local subspace skyline computation

Input: U denotes the query dimensions
$SKY_U \leftarrow \{\emptyset\}$
$threshold \leftarrow$ MAX_INT
$p \leftarrow$ next point based on $f(p)$
while ($f(p) < threshold$) **do**
  if p is not dominated by any point in SKY_U based on U **then**
    remove from SKY_U the points dominated by p
    $SKY_U \leftarrow SKY_U \cup \{p\}$
    $threshold \leftarrow min_{p_i \in SKY_U}(dist_U(p_i))$
  end if
  $p \leftarrow$ next point
end while
return SKY_U

Algorithm 2 Super-peer merging of subspace skylines

Input: U denotes the query dimensions
$SKY_U \leftarrow \{\emptyset\}$
$threshold \leftarrow$ MAX_INT
$SKY_{U_1}...SKY_{U_{Nsp}}$ the super-peers' set of local subspace skyline points
$SKY_{U_s} \leftarrow$ the list with the minimum first element
$p \leftarrow$ next point based on SKY_{U_s}
while ($f(p) < threshold$) **do**
  if p is not dominated by any point in SKY_U on subspace U **then**
    remove from SKY_U the points dominated by p
    $SKY_U \leftarrow SKY_U \cup \{p\}$
    $threshold \leftarrow min_{p_i \in SKY_U}(dist_U(p_i))$
  end if
  $SKY_{U_s} \leftarrow$ the list with the minimum first element
  $p \leftarrow$ next point based on SKY_s
end while
return SKY_U

5.2. Threshold-Based Skyline Computation

Since the acquisition of global knowledge is infeasible in P2P networks, a basic concept of our algorithm is to locally evaluate as many parts of the query as possible. A querying super-peer hands on the query to its neighboring super-peers along the super-peer backbone, which in turn forward the query to their adjacent super-peers, without having to maintain full query processing information. The super-peers execute the query over their local data and retrieve matching points. In the sequel we present all relevant steps in detail.

Each peer executes a local subspace skyline computation (5.2.1). The initiator peer computes the overall subspace skyline result by merging the local results (5.2.2). Finally, we present optimization techniques to reduce the communication and computation cost (5.2.3).

5.2.1 Local Subspace Skyline Computation

Each super-peer maintains a list with the ext-skyline points of its associated peers. We assume that each super-peer can access the stored ext-skyline points in an ascending order of their $f(p)$ values and we present an algorithm for efficient local subspace skyline computation (Algorithm 1). Note that the $f(p)$ value is computed once based on the space D while $dist_U(p)$ refers to the queried subspace U and it is calculated during the skyline computation. During the subspace skyline computation the dominating points among the data already examined are inserted into the current subspace skyline set SKY_U. The algorithm uses as a $threshold$ the minimum value of the $dist_U(p)$ of all points in SKY_U. Based on Observation 5, the algorithm terminates when $threshold$ is smaller than the $f(p)$ value of the next point p.

An important performance parameter is the efficiency of the dominance test, which is computationally expensive if the skyline set contains a large number of points and the dimensionality of the query is high. Notice that we need to test a point for domination only against the potential subspace skyline points SKY_U that have been already found. To speed up the computation the dominance test is performed in a way similar to traditional window queries [14] using a main-memory R-tree with dimensionality equal to the query dimensionality.

5.2.2 Merging of Computed Subspace Skylines

After the initiator super-peer P_{init} has executed a local subspace skyline computation and has collected the local subspace skyline result set of all other super-peers, P_{init} merges the local result sets of the individual super-peers to one global result set. Algorithm 2 performs this task and prunes out those points that are dominated by points of other super-peers SP_j.

We assume that each super-peer delivers its local result set as a sorted list based on the $f(p)$ values of the local skyline points. Again the points are accessed based on their $f(p)$ values while dominated points are discarded, until the $f(p)$ value of the next point is larger than the threshold value. Notice that an alternative would be to merge all lists into one list, sort this list and then apply Algorithm 1, however the use of Algorithm 2 avoids these extra costs and ensures that each list is accessed only until its next element is larger than the threshold value.

5.2.3 Optimization Techniques

As we have already mentioned, each super-peer calculates its own local subspace skyline result. In a second step,

these local subspace skyline results have to be merged into a global result set, for example by the initiator super-peer. Even though the use of local skyline query results at each super-peer reduces the amount of data transmitted to the query initiator P_{init} (naive approach), there is still a chance that a local result may contain points that do not belong to the overall skyline.

We identify two interesting improvements of the basic algorithm. Firstly, our local skyline computation algorithm is threshold-based. Let t be the threshold value at the end of the local skyline computation at the initiator. Based on Observation 5, the threshold value indicates that there is a local skyline point p_{sky} that dominates all points with $f(p)$ values larger that the threshold t. At the end of the local skyline computation the threshold value corresponds to the point with the minimum threshold value, i.e. the highest pruning capability. Since the data is horizontally partitioned over the super-peers the local skyline point p_{sky} dominates all points with a $f(p)$ value larger that the threshold t of all super-peers. Therefore, the t value is attached to the query and is used to further reduce the computation and communication cost, so that the threshold value is propagated to the super-peers and used as an initial threshold value. Consider for example, a super-peer holding points that based on the queried subspace lie near the maximum corner. If there is a small initial threshold value, the local subspace skyline computation ends immediately without returning any points, otherwise the local subspace skyline points are computed and sent back to P_{init}. Thus, the super-peer P_{init} first executes the local subspace skyline computation to obtain an initial value for t, and then the query is forwarded to other super-peers.

A straightforward variation of the above approach is to compute and refine the threshold on each super-peer that is processing the query, instead of forwarding P_{init}'s fixed threshold value. Intuitively, by progressively lowering the threshold value, the pruning capability of the query would be increased at each forwarding step. This approach requires that the query is propagated only after the super-peer has finished the local skyline computation.

A second improvement applies to the merging phase, which can be performed progressively during query evaluation. Instead of forwarding all results back to P_{init}, each super-peer merges the results of its neighbors, and forwards the merged result back to P_{init}. The benefit is twofold. The transferred data is reduced and a time-consuming centralized merging phase is avoided.

Variant	Mnemonic
Fixed Threshold Fixed Merging	FTFM
Fixed Threshold Progressive Merging	FTPM
Refined Threshold Fixed Merging	RTFM
Refined Threshold Progressive Merging	RTPM

Table 2. Variants of the basic SKYPEER.

To summarize, in Table 2 we present our variants of the basic SKYPEER based on two major subspace skyline computation optimization criteria:

1. Threshold propagation: *(i) Fixed Threshold:* P_{init} calculates its threshold t for $q(U,t)$ and forwards the threshold value to all super-peers. *(ii) Refined Threshold:* P_{init} calculates and sends its threshold to its neighboring super-peers, which do not forward it immediately to other super-peers, but rather they first compute the subspace skyline, calculate the new threshold t', and then forward $q(U,t')$.

2. Merging strategy: *(i) Fixed Merging at* P_{init}*:* In this approach, all super-peers forward their computed subspace skyline back to P_{init}, and P_{init} is responsible for merging the results and computing the resulting subspace skyline for $q(U)$. *(ii) Progressive Merging:* Each super-peer merges the results it receives with its locally computed subspace skyline, before sending the results back to the super-peer from which it originally received the query.

We now introduce the SKYPEER algorithm, which runs on each super-peer SP_i. A subspace skyline query $q(U)$ is posed by the initiator super-peer P_{init}. The initiator super-peer first computes the skyline on its local ext-skyline $\cup$ $ext\text{-}SKY_{D_i}$ $(i = 0..DEG_p)$. This results in a threshold value t, which based on Observation 5 can be used to prune out points that cannot belong to the result of the skyline query. The threshold value is attached to the query $q(U)$ that becomes $q(U,t)$ before it is forwarded to SP_{init}'s neighbors at super-peer level. Each super-peer receiving the query, forwards it to its neighboring super-peers and executes a local *threshold-based* subset skyline calculation on its set of local ext-skyline points. If the RT*M variants are employed, then before forwarding the query, SP_i computes the skyline on its local ext-skyline which results in a refined threshold value t' (or in worst case $t' = t$) that is attached to the query before it is send to the neighboring super-peers. The results are routed back to P_{init}. If one of the *TPM variants is employed, SP_i first merges the results it receives, before forwarding them to the peer from which it received the query.

Correctness of the algorithm: Observation 4 guarantees that there are no false negatives caused by the preprocessing phase. Observation 5 ensures that there are no false negatives during the local skyline computation and the merge phase, in contrast to the domination test that ensures that there are no false positive. Since there are no false negatives nor false positives, SKYPEER computes the exact subspace skyline results.

Algorithm 3 SKYPEER on super-peer SP_i

Input: mode (FTFM, FTPM, RTFM, RTPM)
Query q on subspace U with threshold t ($q(U,t)$)
Peer from which q was received (SP_q)
$L_N \leftarrow$ list of all neighbors except SP_q
if (mode $\in$ {RTFM,RTPM}) or ($SP_i = P_{init}$) **then**
  $SKY_{U_0} \leftarrow$ compute local skyline
  $t \leftarrow refineThreshold(t)$
end if
for $n_i \in L_N$ **do**
  send(n_i, $q(U,t)$)
end for
if mode $\in$ {FTFM,FTPM} **then**
  $SKY_{U_0} \leftarrow$ compute local skyline
end if
for $j = 1$ **to** $|L_N| - 1$ **do**
  receive SKY_{U_j}
  if mode $\in$ {FTPM,RTPM} **then**
    $SKY_{U_0} \leftarrow mergeResults(SKY_{U_j}, SKY_{U_0})$
  else
    send(SP_q, SKY_{U_j})
  end if
end for
send(SP_q, SKY_{U_0})

5.3. Pre-processing Phase

In the pre-processing phase, each peer P_i calculates its own ext-skyline $ext\text{-}SKY_{D_i}$ in the original space D and sends it to the associated super-peer. Despite the fact that any of the existing centralized skyline algorithms may be applied to calculate the ext-skyline, if the domination test is replaced by the ext-domination definition, in our example we assume that Algorithm 1 is used.

The task of the super-peer is to merge the local ext-skyline of the individual peers to one result set that constitutes the ext-skyline of space D with respect to the dataset on the super-peer and its associated peers. Algorithm 2 merges the lists and prunes out those points of a peer P_i that are ext-dominated by points of another peer P_j.

We illustrate the details of peer pre-processing by means of an example. In Figure 2, three peers (P_A, P_B, P_C) assigned to super-peer SP_A and their local datasets are shown. The dimensionality of the dataset is 4. Each peer computes its local ext-skyline in the original space. The points added to the result set due to the ext-skyline definition are grey shaded. For instance, four of the five points of P_A are skyline points, while $A3$ is included as an ext-skyline point. Similarly, for P_C the skyline point is $C4$, while the ext-skyline points are $C4$ and $C5$.

In the case that a new peer P_J joins the network, it is associated with a super-peer using the basic bootstrapping protocol. If the super-peer has already executed an ext-skyline computation, so it has available a list of ext-skyline results of all other associated peers, the ext-skyline of P_J must be merged with the existing ext-skyline at the super-peer. Notice that this merging is performed incrementally, i.e. there is no need to process again all the lists of ext-skyline points from all associated peers, so the additional processing cost of peer joins is very low. We intend to address peer failures in our future work.

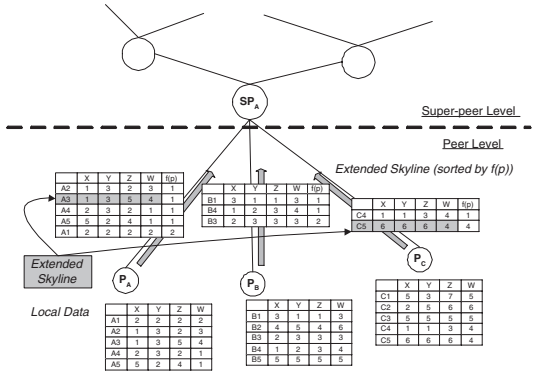


Figure 2. Peer pre-processing example.

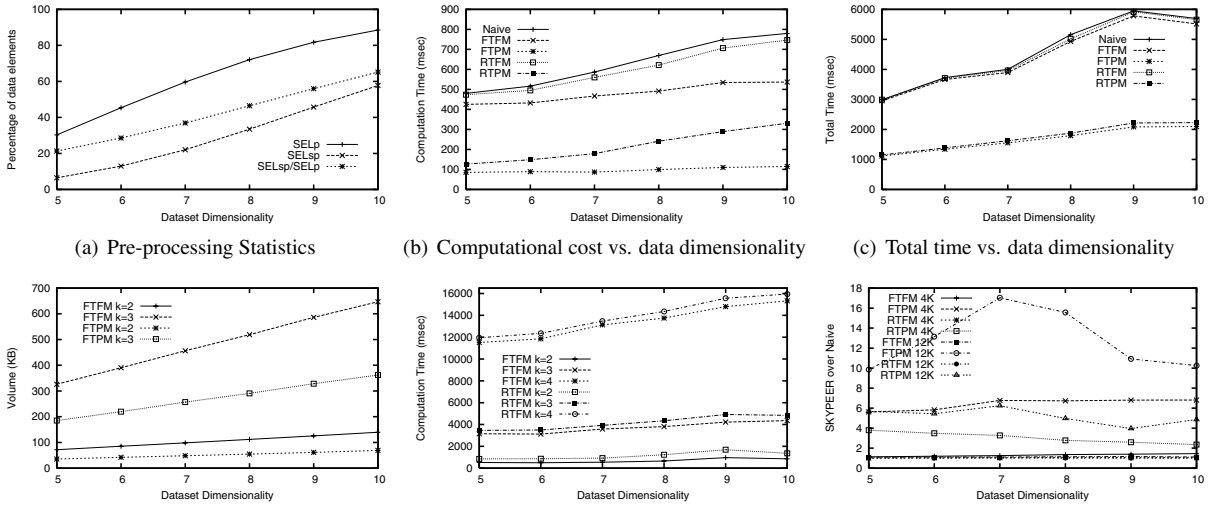
6. Experimental Evaluation

We evaluate the performance of the SKYPEER algorithm which was implemented in Java. The simulations run on 3GHz Pentium PCs and the data was stored locally. In order to be able to test the algorithms with realistic network sizes, we ran multiple instances of the peers on the same machine and simulated the network interconnection.

The P2P network topology used in the experiments consists of N_{sp} interconnected super-peers in a random graph topology. We used the GT-ITM topology generator [3] to create well-connected random graphs of N_{sp} peers with a user-specified average connectivity (DEG_{sp}). In our experiments we vary the network size (N_p) from 4000 to 80000 peers, while the number of super-peers is $N_{sp} = 5\% \times N_p$ (for $N_p \geq 20000$ we used $N_{sp} = 1\% \times N_p$). We also tested different DEG_{sp} values ranging from 4 to 7.

We used synthetic data collections and query workloads. The dataset was horizontally partitioned evenly among the peers. We used two different datasets: uniform and clustered. The uniform dataset includes random points in a unit space. For the clustered dataset, each super-peer picks cluster centroids randomly and all associated peers obtain points, the coordinates of which follow a Gaussian distribution on each axis with variance 0.025, and a mean equal to the corresponding coordinate of the centroid. We conduct experiments varying the dimensionality (5-10) and the

[3] Available at: http://www.cc.gatech.edu/projects/gtitm/



(a) Pre-processing Statistics (b) Computational cost vs. data dimensionality (c) Total time vs. data dimensionality

(d) Volume of transferred messages (e) Scalability with k, for FTFM and RTFM (f) SKYPEER's relative performance to naive

Figure 3. Comparison of SKYPEER variants for different data dimensionality values.

cardinality (1M-20M points) of the dataset. Given a query dimensionality, all dimension subsets have uniform probability to be requested. We generate 100 queries, and for each query a super-peer initiator is randomly selected. We measure the average: (i) skyline query processing computational time, (ii) total time (including network delay), and (iii) volume of transferred data, for all queries. Unless mentioned explicitly, we use the following default values: $d = 8$, $k = 3$, $DEG_{sp} = 4$, $N_p = 4000$, each peer holds 250 points and the dataset is uniform.

6.1. Pre-processing Statistics

The smaller the size of the ext-skyline, the larger the savings that we obtain in the pre-processing step. In the first experiment we consider a uniformly distributed data set over 4000 peers, while varying d. Figure 3(a) illustrates the percentage of the points that belong to the ext-skyline. The peer selectivity (SEL_p) indicates the portion of the data points transmitted from the peers to the super-peers, i.e. the average number of local ext-skyline points. The super-peer selectivity (SEL_{sp}) represents the average number of ext-skyline points for each super-peer as a percentage of all the data points. We also show the percentage of data points that are ext-skyline points at a super-peer, in comparison to the total number of ext-skyline points of the associated peers, denoted (SEL_{sp}/SEL_p).

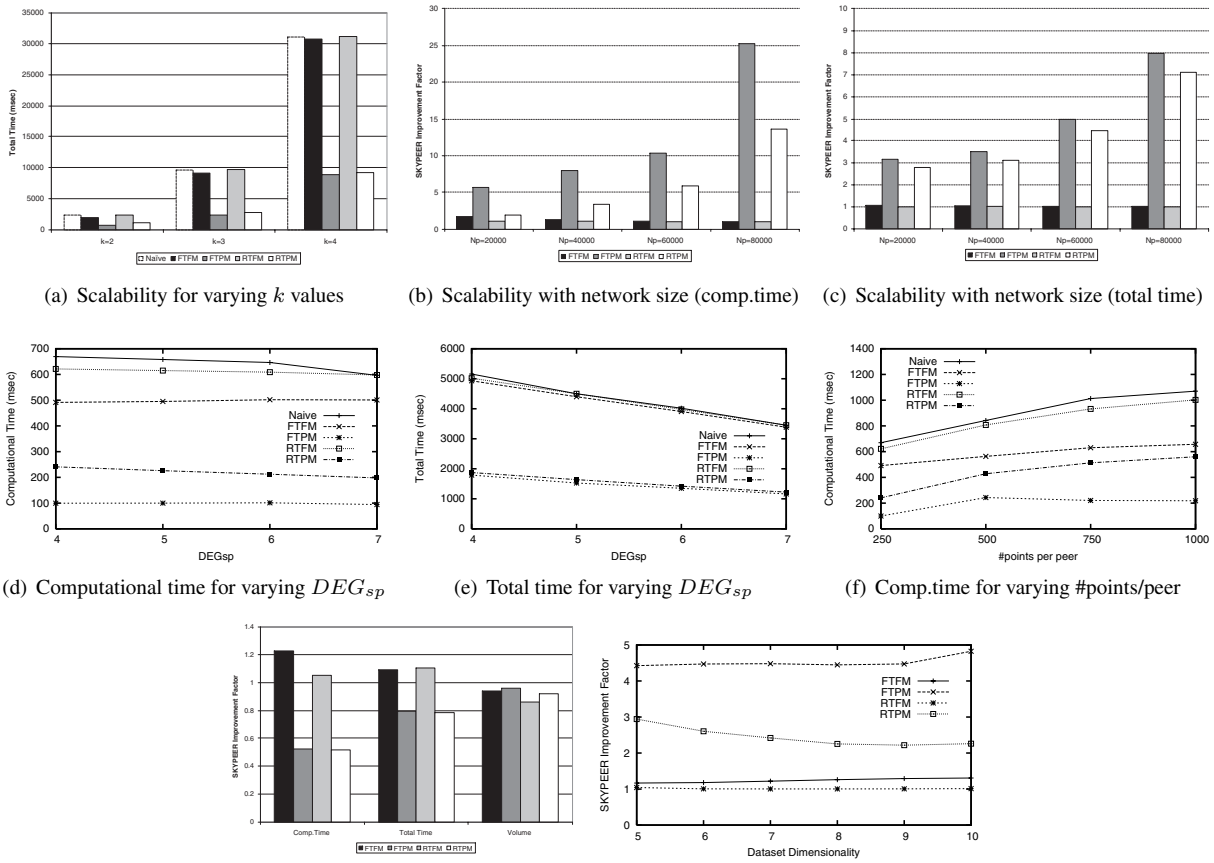
For example for $d = 7$ dimensions, 59% of the data points are transmitted to the super-peers, while only a 22% actually belong to the union of all local ext-skylines at super-peer level. Clearly, for smaller values of data dimensionality, only a small portion of the points are used, thus we can prune a significant number of points during the pre-processing step. As only the ext-skyline points are stored in the super-peers, the number of data points processed at query time is a small percentage of the whole dataset, keeping our approach feasible.

6.2. Scalability Study

In Figure 3(b), we present the performance of the SKYPEER framework in terms of computational time, neglecting network delays. The results show that the refined threshold variants (RT*M) are more costly than the fixed threshold variants (FT*M), however they are still more efficient than the naive one, because of threshold usage. Further, progressive merging (*TPM) is faster than fixed merging (*TFM) at the initiator, because the computational cost to merge all N_{sp} lists at the initiator is higher compared to the total cost incurring at intermediate super-peers.

The plot in Figure 3(c) illustrates the total response time taking into account the network delay, which depends on the size of transmitted data. We assume 4KB/sec as the network transfer bandwidth on each connection. This chart shows that progressive merging (*TPM) keeps the total response time low. In all cases the four variants of SKYPEER constantly outperform the naive algorithm. In Figure 3(d), the volume of the messages (in KB) is illustrated as a function to the dataset dimensionality for the uniform dataset. The query dimensionality varies varies from 2 to 3 and two different variations of SKYPEER are examined. The volume of the messages exchanged to retrieve the skyline result depends mainly on the merging strategy. We present the results of FTFM and FTPM as representative ones. Obviously progressive merging reduces the volume of data transferred in the network.

In the next series of experiments we examine the proposed method's scaling features with regards to query di-



(a) Scalability for varying k values (b) Scalability with network size (comp.time) (c) Scalability with network size (total time)

(d) Computational time for varying DEG_{sp} (e) Total time for varying DEG_{sp} (f) Comp.time for varying #points/peer

(g) SKYPEER vs. naive, clustered dataset (h) SKYPEER vs. naive, clustered dataset

Figure 4. Scalability study of SKYPEER for different k, N_p, DEG_{sp} and number of points per peer.

mensionality. In order to study SKYPEER's behavior for larger networks, we increased the network size from 4000 to 12000 peers. Figure 3(e) compares the performance of the fixed (FTFM) against the refined (RTFM) threshold variants for fixed merging. We measured the computational time needed for calculating the subspace skyline, for different query dimensionality values ranging from 2 to 4. We conclude that the fixed threshold variants perform better than the refined threshold ones, however this is due to the uniform dataset employed, which does not allow the exploitation of refining the threshold value. Then, in Figure 3(f), we present a performance comparison between SKYPEER and the naive algorithm for different network sizes. The vertical axis represents the relative performance of SKYPEER as compared to the naive approach. It is clear that SKYPEER always outperforms naive, and for large networks (12000 peers) the FTPM variant is 17 times faster than naive.

Thereafter, we examined the performance of our algorithm depending on the network topology, in terms of participating peers number and connectivity degree. We tested the scalability features of SKYPEER with k for a network of 12000 peers (Figure 4(a)). We measured the total response time of all algorithms for different query dimensionality. It is obvious that progressive merging scales much better with k, providing an efficient solution as compared to the naive one.

Figures 4(b) (computational time) and 4(c) (total time) show the comparative performance of SKYPEER variants against naive, when the network size is increased from 20000 to 80000 peers. The improvement factor of progressive merging (*TPM) over the naive approach increases with network size, proving the scalability of our approach.

In the next experiment, we study how different super-peer connectivity degrees (DEG_{sp}) affect the performance of SKYPEER. In Figures 4(d) and 4(e), we experiment with a network of 4000 peers with DEG_{sp} varying from 4 to 7. We conclude that even though the computational time is not affected by DEG_{sp}, the total time is reduced when DEG_{sp} increases. This is because higher DEG_{sp} values, result in smaller routing paths, hence smaller network transfer costs.

In Figure 4(f), we increased the number of points per peer (n/N_p) from 250 to 1000. We notice that the progressive merging variants clearly outperform the fixed merging ones, as the number of points per peer increases.

We also evaluate SKYPEER on a clustered dataset and the results are shown in Figure 4(g). In this experiment,

we generated a clustered 3-dimensional dataset and we used $k = 3$, in order to study the effect of subspace skylines on a clustered subspace. We use global skyline queries to avoid distortion of the clustered data distribution through the projection. As expected fixed threshold variants perform better w.r.t. computational time, however when it comes to total time, the refined threshold variants are better. This experiments indicates the usefulness of RT*M variants, when the dataset is clustered, and when the first priority is to keep the network load low. In a similar experiment, in Figure 4(h), we see that RT*M variants perform better also for increasing dataset dimensionality values. Again the importance of refining threshold is elevated, when the dataset is clustered.

As expected the fixed threshold method achieves a fast overall response time. This is because, ideally, the query will reach all super-peers in the minimum possible time. Then they can immediately start the local skyline processing and return their results. It was expected that the refined threshold would lead to query pruning capability improvements, and therefore to reduce processing costs in super-peers for skyline computation and communication costs. Unexpectedly in most experiments there is no noticeable benefit. This is mainly because of the nature of the synthetic dataset, i.e. uniform dataset distributed uniformly among the peers. However in the case of clustered datasets, we see that threshold refinement leads to performance gains.

Progressive merging reduces communication costs, since some candidate skyline points are pruned out earlier. The computational cost is further reduced through the progressive merging since the fixed merging at P_{init} is very costly because of the high number of elements. Thus, the computational cost of merging is distributed over many super-peers, instead of only to P_{init} and thus a potential bottleneck at P_{init} is avoided.

7. Conclusions

In this paper we addressed the issue of subspace skyline computation in a P2P setting. To the best of our knowledge this is the first paper to confront the combination of (i) large degree of data distribution and (ii) subspace skyline queries. We propose a threshold based algorithm, called SKYPEER, which forwards the skyline query requests among peers, in such a way that the amount of data to be transferred is significantly reduced. Additionally, the extended skyline operator, which is applicable to all skyline algorithms, provides an effective solution for retrieving a superset of all subspace skyline results. Finally our experimental evaluation shows that our algorithm, SKYPEER, is much more efficient than the baseline approach, both in terms of computational and communication costs. In our future work we will investigate how churn, in particular peer failure, affects the performance of SKYPEER.

References

[1] W.-T. Balke, U. Gunzer, and J. Zheng. Efficient distributed skylining for web information systems. In *Proceedings of EDBT'04*, pages 256–273, 2004.

[2] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. In *Proceedings of ICDE'05*, 2005.

[3] I. Bartolini, P. Ciaccia, and M. Patella. SaLSa: Computing the skyline without scanning the whole sky. In *Proceedings of CIKM'06*, 2006.

[4] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedigns of ICDE'01*, pages 421–430, 2001.

[5] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with pre-sorting. In *Proceedings of ICDE'03*, pages 717–719, 2003.

[6] E. Dellis, A. Vlachou, I. Vladimirskiy, B. Seeger, and Y. Theodoridis. Constrained subspace skyline computation. In *Proceedings of CIKM'06*, 2006.

[7] C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. DESENT: Decentralized and distributed semantic overlay generation in P2P networks. *To appear in IEEE Journal on Selected Areas in Communications (J-SAC)*, 2007.

[8] K. Hose, M. Karnstedt, A. Koch, K. Sattler, and D. Zinn. Processing rank-aware queries in P2P systems. In *Proceedings of DBISP2P'05*, pages 238–249, 2005.

[9] K. Hose, C. Lemke, and K.-U. Sattler. Processing relaxed skylines in PDMS using distributed data summaries. In *Proceedings of CIKM'06*, 2006.

[10] Z. Huang, C. Jensen, H. Lu, and B.-C. Ooi. Skyline queries against mobile lightweight devices in MANETs. In *Proceedings of ICDE'06*, 2006.

[11] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *Proceedings of VLDB'02*, pages 275–286, 2002.

[12] S. Michel, P. Triantafillou, and G. Weikum. KLEE: a framework for distributed top-k query algorithms. In *Proceedings of VLDB'05*, pages 637–648, 2005.

[13] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loeser. Super-peer-based routing and clustering strategies for RDF-based P2P networks. In *Proceedings of WWW'03*, 2003.

[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1):41–82, 2005.

[15] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *Procedings of VLDB'05*, pages 253–264, 2005.

[16] K. Tan, P. Eng, and B.-C. Ooi. Efficient progressive skyline computation. In *Proceedings of VLDB'01*, pages 301–310, 2001.

[17] Y. Tao, X. Xiao, and J. Pei. SUBSKY: Efficient computation of skylines in subspaces. In *Proceedings of ICDE'06*, 2006.

[18] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi. Parallelizing skyline queries for scalable distribution. In *Proceedings of EDBT'06*, pages 112–130, 2006.

[19] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of ICDE'03*, pages 49–, 2003.

[20] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *Proceedings of VLDB'05*, pages 241–252, 2005.