Efficient Distributed Skylining for Web Information Systems

Wolf-Tilo Balke[1], Ulrich Güntzer[2], and Jason Xin Zheng[1]

[1] Computer Science Department, University of California,
Berkeley, CA 94720, USA
{balke,xzheng}@eecs.berkeley.edu
[2] Insitut für Informatik, Universität Tübingen,
72076 Tübingen, Germany
guentzer@informatik.uni-tuebingen.de

Abstract. Though skyline queries already have claimed their place in retrieval over central databases, their application in Web information systems up to now was impossible due to the distributed aspect of retrieval over Web sources. But due to the amount, variety and volatile nature of information accessible over the Internet extended query capabilities are crucial. We show how to efficiently perform distributed skyline queries and thus essentially extend the expressiveness of querying today's Web information systems. Together with our innovative retrieval algorithm we also present useful heuristics to further speed up the retrieval in most practical cases paving the road towards meeting even the real-time challenges of on-line information services. We discuss performance evaluations and point to open problems in the concept and application of skylining in modern information systems. For the curse of dimensionality, an intrinsic problem in skyline queries, we propose a novel sampling scheme that allows to get an early impression of the skyline for subsequent query refinement.

1 Introduction

In times of the ubiquitous Internet the paradigm of Web information systems has substantially altered the world of modern information acquisition. Both in business and private life the support with information that is stored in a decentralized manner and assembled at query time, is a resource that users more and more rely on. Consider for instance Web information services accessible via mobile devices. First useful services like city guides, route planning, or restaurant booking have been developed [5], [2] and generally all these services will heavily rely on information distributed over several Internet sources possibly provided by independent content providers. Frameworks like NTT DoCoMo's i-mode [18] already provide a common platform and business model for a variety of independent content providers.

Recent research on web-based information systems has focused on employing middleware algorithms, where users had to specify weightings for each aspect of their query and a central compensation function was used to find the best matching objects [7], [1]. The lack of expressiveness of this 'top k' query model, however, has first been addressed by [8] and with the growing incorporation of user preferences into

E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992, pp. 256–273, 2004.

database systems [6], [10] and information services [22] the limitations of the entire model became more and more obvious. This led towards the integration of so-called 'skyline queries' (e.g. [4]) into database systems. Basically the 'skyline' is a non-discriminating combination of numerical preferences under the notion of Pareto optimality. Since it was only proposed for database systems working over a central (multi-dimensional) index structure, extending its expressiveness also to the broad class of Web information systems is most desirable. The contribution of this paper is to undertake this task and present an efficient algorithm with proven optimality. We will present a distributed skylining algorithm and show how to enhance its efficiency for most practical cases by suitable heuristics. We will also give an extensive performance evaluation and propose a scheme to cope with high-dimensional skylines.

As a running example throughout this paper we will focus on a typical Web information service scenario. Our algorithm will support a sample user interacting with a route planning service like e.g. Map-Quest's Road Trip Planner or Driving Directions [16]. This is a characteristic example of a Web service where the gathering of on-line information is tantamount: though a routing service is generally capable of finding possible routes, the quality of certain routes - and thus their desirability to the user - may heavily differ depending on current information like road blockings, traffic jams or the weather conditions. Thus we first have to collect a set of user-specified preferences and integrate them with our query to the routing system. But of course users won't be able to specify something like 'for my purposes the shortest route is 0.63 times more important than that there is no jam' in a sensible, i.e. *intuitive*, way. Queries rather tend to be formulated like 'I would prefer my route to be rather short and with little jams' giving no explicit weightings for a compensation function. Hence the *skyline* over the set of possible routes is needed for a high quality answer set. Since there often are many sources on the Internet offering current traffic information, usually the query also will have to be posed to a variety of sources needing an efficient algorithm for the distributed skyline computation. The example of a route planning service also stresses the focus on real time constraints, because most on-line information like traffic jams or accidents will have to be integrated on the fly and delivered immediately to be of use for navigation. Since such efficient algorithms for distributed retrieval are still problematic, today's web portals like Map-Quest allow only a minimum of additional information (e.g. avoiding toll roads) and use central databases, that provide necessary information. However, given the dynamic nature of the Web this does not really meet the challenges of Web information systems.

2 Web Information Systems Architecture and Related Work

Modern Web information systems feature an architecture like the one roughly sketched in figure 1. Using a (mobile) client device the user poses a query. Running on an application/Web server this query may be enriched with information about a user (e.g. taken from stored profiles) and will be posed to a set of Internet sources. Depending on the nature of the query different sources can be involved in different parts of the query, e.g. individual sources for traffic jams or weather information. Collecting the individual results the combining engine runs an algorithm to compute

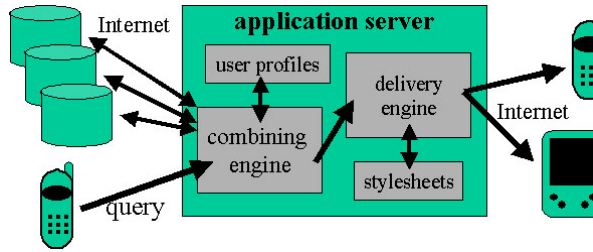

Fig. 1. Web information system architecture

the overall best matching objects. These final results have then to be aggregated according to each individual user's specifications and preferences. After a transformation to the appropriate client format (e.g. using XSLT with suitable stylesheets) the best answers will be returned to the user.

The first area to address such a distributed retrieval problem was the area of 'top k retrieval' over middleware environments, e.g. [7], [9], [19]. Especially for content-based retrieval of multimedia data these techniques have proven to be particularly helpful. Basically all algorithms distinguish between different query parts (subqueries) evaluating different characteristics, which often have to be retrieved from various subsystems or web sources. Each subsystem assesses a numerical score value (usually normalized to [0,1]) to each object in the collection. The middleware algorithms use two basic kinds of accesses that can be posed: there is the iteration over the best results from one source (or with respect to a single aspect of the query) called a '*sorted access*' and there is the so-called '*random access*' that retrieves the score value with respect to one source or aspect for a certain given object.

The physical implementation of these accesses always strongly depends on the application area and will usually differ from system to system. The gain of speeding up a single access (e.g. using a suitable index) will of course complement the total run-time improvement by reducing the overall number of accesses. Therefore minimizing the number of necessary object accesses and thus also the overall query runtimes is tantamount to build practical systems (with real-time constraints) [1]. Prototypical Web information systems of that kind are e.g. given by [3], [5] or [2]. However, all these top k retrieval systems relied on a single combining function (often called 'utility function') that is used to compensate scores between different parts of the query. Being worse in one aspect can be compensated by the object doing better in another part. However, the semantic meaning of these (user provided) combining functions is unclear and users often have to guess the 'right' weightings for their query. The area of operations research and research in the field of human preferences like [6] or [8] has already since long criticized this lack in expressiveness.

A more expressive model of non-discriminating combination has been introduced into the database community by [15]. The 'skyline' or 'Pareto set' is a set of *non-dominated* answers in the result for a query under the notion of Pareto optimality. The typical notion of Pareto optimality is that without knowing the actual database content, there can also be no precise a-priori knowledge about the most sensible optimization in each individual case (and thus something that would allow a user to choose weightings for a compensation function). The Pareto set or skyline hence contains all best matching object for all possible strictly monotonic optimization

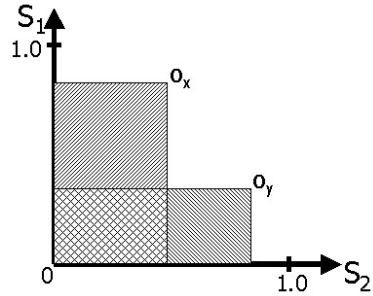

Fig. 2. Skyline objects and regions of domination

functions. An example for skyline objects with respect to two query parts and their scorings S_1 and S_2 is shown in figure 2. Each database object is seen as a point in multidimensional space characterized by its score values. For instance objects o_x =(0.9, 0.5) and o_y=(0.4, 0.9) both dominate all objects within a rectangular area (shaded). But o_x and o_y are not comparable, since o_x dominates o_y in S_1 and o_y dominates o_x in S_2. Thus both are part of the skyline.

Whereas [15] and the more recent extensive system in [12] with an algebra for integrating the concept of Pareto optimality with the top k retrieval model for preference engineering and query optimization in databases [11], are more powerful in that they do not restrict skyline queries to numerical domains, they both rely on the naïve algorithm of quadratic complexity doing pairwise comparisons of all database objects. Focusing on numerical domains [4] was able to gain logarithmic complexity along the lines of [14]. Initially skyline queries were mainly intended to be performed within a single database query engine. Thus the first algorithms and subsequent improvements all work on a central (multidimensional) index structure like R*-trees [20], certain partitioning schemes [21] or k-nearest-neighbor searches [13]. However, such central indexes cannot be applied to distributed Web information systems. Since there is still no algorithm to process distributed skyline queries, up to now the extension of expressiveness of the query model could not be integrated in Web information services. We will deal with the problem of designing an efficient distributed algorithm for computing skyline queries only relying on sorted and random accesses.

3 A Distributed Skylining Algorithm

In this section we will investigate distributed skylining and present a first basic algorithm. As we have motivated in the previous section the basic skyline consists of all non-dominated database objects. That means all database objects for which there is no object in the database that is better or equal in all dimensions, but in at least one aspect strictly better. Assuming every database object to be represented by a point in n-dimensional space with the coordinates for each dimension given by its scores for the respective aspect, we can formulate the problem as:

The Skyline Problem: Given set $O := \{o_1,\ldots,o_N\}$ of N database objects, n score-functions $s_1,\ldots s_n$ with $s_i : O \rightarrow [0,1]$ and n sorted lists $S_1,\ldots,S_n$ containing all database objects and their respective score values using one of the score function for each list; all lists are sorted descending by score values starting with the highest scores. Wanted is the subset P of all non-dominated objects in O, i.e. $\{o_i \in P \mid \neg\exists\ o_j \in O : (s_1(o_i) \leq s_1(o_j) \wedge \ldots \wedge s_n(o_i) \leq s_n(o_j) \wedge \exists\ q \in [1,\ldots,n] : s_q(o_i) < s_q(o_j)) \}$

We will now approach a suitable distributed algorithm to efficiently find this set. Our algorithm basically consists of three phases: The first phase (step 1) will perform sorted accesses until we have definitely seen all objects that can possibly be part of the skyline. The second phase (step 2 and 3) will extend the accesses on all objects with minimum seen scores in the lists and will prune all other database objects. The third phase (step 4) will employ focused random accesses to discard all seen objects that are dominated before returning the skyline to the user. To keep track of all accessed objects we will need a central datastructure containing all available information about all objects seen, but also group the objects with respect to the sorted lists that they have occurred in. The beauty of this design is that we only have to check for domination within the small sets for each list and can return some first results early.

Basic Distributed Skyline Algorithm

0. Initialize a datastructure $P := \emptyset$ containing records with an identifier and n real values indexed by the identifiers, initialize n lists $K_1,\ldots,K_n := \emptyset$ containing records with an identifier and a real value, and initialize n real values $p_1,\ldots,p_n := 1$

1. Initialize counter i := 1.

1.1. Get the next object o_{new} by sorted access on list S_i

1.2. If $o_{new} \in P$, update its record's i-th real value with $s_i(o_{new})$, else create such a record in P

1.3. Append o_{new} with $s_i(o_{new})$ to list K_i

1.4. Set $p_i := s_i(o_{new})$ and $i := (i \bmod n) + 1$

1.5. If all scores $s_j(o_{new})$ $(1 \leq j \leq n)$ are known, proceed with step 2 else with step 1.1.

2. For i = 1 to n do

2.1. While $p_i = s_i(o_{new})$ do sorted access on list S_i and handle the retrieved objects like in step 1.2 to 1.3

3. If more than one object is entirely known, compare pairwise and remove the dominated objects from P.

4. For i = 1 to n do

4.1. Do all necessary random accesses for the objects in K_i that are also in P, immediately discard objects that are not in P

4.2. Take the objects of K_i and compare them pairwise to the objects in K_i. If an object is dominated by another object remove it from K_i and P

5. Output P as the set of all non-dominated objects

For ease of understanding we show how the algorithm works for our running example: for mobile route planning in [2] we have shown for the case of top k retrieval how traffic information aspects can be queried from various on-line sources. Posing a query on the best route with respect to say its length (S_1) and the traffic density (S_2) our user employs functions that evaluate the different aspects, but is not sure how to compensate length and density. The following tables show two result lists with some routes R_i ordered by decreasing scores with respect to their length and current traffic density:

S_1 (length)					
R1	R3	R5	R4	R7	…
0.9	0.9	0.8	0.8	0.7	…

S_2 (traffic density)					
R2	R4	R6	R3	R8	…
0.9	0.8	0.8	0.8	0.7	…

The algorithm in step 1 will in turn perform sorted accesses on both lists until the first route R4 has been seen in both lists leading to the following potential skyline objects:

Route	R1	R2	R3	**R4**	R5	R6
Score S_1	0.9	?	0.9	**0.8**	0.8	?
Score S_2	?	0.9	?	**0.8**	?	0.8

In step 2 we will do some additional sorted accesses on all routes that possibly could also show the current minimum score in each list and find that R7 in S_1 already has a smaller score, hence we can discard it. In contrast R3 in S_2 has the current minimum score, hence we have to add it to our list, but can then discard the next object R8 in S_2, which does have a lower score. Step 3 now tests, if one of the two completely seen routes R3 and R4 is dominated: by comparing their scores we find that R4 is dominated by R3 and can thus be discarded. We can now regroup objects into sets K_i and do all necessary random accesses and the final tests for domination only within each set.

K_1		
R1	R3	R5
0.9	0.9	0.8
?	0.8	?

K_2		
R2	R6	R3
?	?	0.9
0.9	0.8	0.8

Step 4 now works on the single sets K_i. We have to make a random access on R1 with respect to S_2 and find that its score is say 0.5. Thus we get its score pair (0.9, 0.5) and have to check for domination within set K_1. Since it is obviously dominated by the score pair (0.9, 0.8) of R3, we can safely discard R1. Doing the same for object R5 we may retrieve a value of say 0.6 thus R5's pair (0.8, 0.6) is also dominated by R3 and we are finished with set K_1. Please note that we could have saved this last random access on R5, since we already know that all unknown scores in S_2 must be smaller than the current minimum of the respective list (in this case 0.8). This would already have shown R5's highest possible score pair (0.8, 0.8) to be dominated by R3. At this point we are already able to output the non-dominated objects of K_1, since lemma 2 shows that if any of the objects of set K_1 should be dominated by objects in another set K_i , they also always would be dominated by an object *in* K_1.

Dealing with K_2 we have to make random accesses for S_1 on routes R2 and R6 and find for route R2 a score value of say 0.6 leading to a score pair of (0.6, 0.9). But it cannot be dominated by R3's pair (0.9, 0.8), as its score in S_2 is higher than R3's. Finally for R6 we may find a score of say 0.2, thus it is dominated by R3 and can be discarded (also in these cases we could have saved two random accesses like shown above). We now can deliver the entire skyline to our user's query consisting of routes R3 and R2. All other routes in the database (either seen or not yet accessed) are definitely dominated by at least one of these two routes.

Independently of any weightings a user could have chosen for a compensation function thus either route R3 or R2 would have turned up as top object dominating all other routes. Delivering them both as top objects saves users from having to state unintuitive a-priori weightings and allows for an informed choice according to each individual user's preferences. But we still have to make sure, that upon termination no pruned object can belong to the skyline and no dominated object will ever be returned. We will state two lemmas and then prove the correctness of our algorithm.

Lemma 1 (Discarding unseen objects)
After an object o_x has been seen in each list and all objects down to at least score $s_i(o_x)$ $(1 \le i \le n)$ in each list have also been seen, the objects not yet seen cannot be part of the skyline, more precisely they are dominated by o_x.

Proof: Let o_x be the object seen in all lists. Then with p_i as the minimum score seen in each list we have due to the sorting of the lists $\forall\ i\ (1 \le i \le n) : s_i(o_x) \ge p_i$. Since all objects having a score of at least score p_i in list i have been collected, we can conclude that any not yet seen object o_{unseen} satisfies $\forall\ i\ (1 \le i \le n) : s_i(o_{unseen}) < p_i \le s_i(o_x)$ and thus $\forall\ i\ (1 \le i \le n) : s_i(o_{unseen}) < s_i(o_x)$. Hence o_{unseen} is dominated by o_x and thus cannot be part of the skyline independently of o_x itself being part of the set or being dominated. ■

We can even show the somewhat stronger result that, if we have seen an object o_x in all lists and stop the sorted accesses in step 2 after seeing only a single worse object in any of the lists, we can still safely discard all unseen objects. This is because we need the strict '<' in one single list only. In the other lists '≤' would still be sufficient (since due to sorting '=' is the highest possible).

Lemma 2 (Objects can only be dominated by objects in the same set K_i)
Assume that all objects that have been seen, are divided into n sets according to the lists in which they occurred, i.e. if an object o_x occurs in list i $(1 \le i \le n)$ it is added to set K_i. Lets further assume that in the lists in which o_x occurred, all objects having at least the respective score value of o_x have also been seen. Then, if the object o_x in any set K_i is dominated by any other object, this object has also to be part of set K_i.

Proof: Let o_x be any dominated object already seen and assigned to at least one set K_i $(1 \le i \le n)$. Due to Lemma 1 o_x cannot be dominated by any unseen object, thus the dominating object o_y has already been seen and thus has also been assigned to at least one of the sets K_i $(1 \le i \le n)$. If o_x and o_y are in exactly the same sets there is nothing to show. Thus let us assume o_y dominates o_x and there exists at least one set K_j $(1 \le j \le n)$ containing object o_x, but not object o_y. Thus due to the sorting of the lists and the fact that we have seen all objects in list x having at least the respective score value $s_j(o_x)$ of object o_x, we have to conclude that $s_j(o_y) < s_j(o_x)$ in contrast to the assumption that o_y dominates o_x. ■

Theorem 1: (Correctness of the Basic Algorithm)
The basic algorithm always terminates and delivers the entire set of non-dominated objects and only the set of non-dominated objects.

Proof: Since the termination is obvious, we have to show that a) no relevant object is missed by our algorithm and that b) no object in the returned set can be dominated by any other object.

Ad a) Steps 1 and 2 of the algorithm collect all objects, until one object has been seen in all lists and all objects of the minimum score in each list have also been seen. Thus Lemma 1 applies and we can safely discard all unseen objects, since they cannot

be part of the skyline. In steps 3 and 4 only dominated objects are discarded (in step 4 we also might use upper boundary estimations of some scores for discarding, but since the upper boundaries are best case estimations, it is obvious that objects discarded in step 4 would also be discarded using their actual score values). Thus the set returned in step 5 will contain all objects of the skyline.

Ad b) After steps 1 to 3 Lemma 2 applies and we can restrict the search for dominating objects to the sets K_i ($1 \leq i \leq n$). Since in any set K_i no object can be dominated by an object having a strictly smaller score with respect to the i-th list, it is sufficient to do pairwise comparisons with those objects having a larger or equal score. Thus Step 4 correctly discards all dominated objects within the sets K_i and since all objects returned in step 5 must have been part of at least one K_i, they cannot be dominated by any object. ■

Since the algorithm is supposed to work with distributed web sources thus having rather high access costs, for the optimality and complexity considerations we have to focus on the necessary object accesses instead of main memory operations. The next theorem will show that the termination condition of phase 1 is optimal, since one of the objects seen in all lists is definitely part of the skyline. Stopping earlier would thus discard possibly non-dominated objects; we therefore have to see an object in all lists.

Theorem 2 (Optimality of Sorted Accesses):
The basic algorithm uses an optimal number of sorted accesses.

Proof: Sorted accesses are only made during the first two steps. After the first step one object has been seen in all lists. In step 2 we do further sorted accesses to get all objects with at least one score equal to the respective minimum score in each list. If we can show that among these objects and the object seen in all lists there always is at least one object belonging to the skyline, we could not stop doing sorted accesses earlier and thus use an optimal number of sorted accesses.

Let o_x be the first object that has occurred in all lists and p_i ($1 \leq i \leq n$) be the minimum scores in each list. Then we get $\forall\, i\ s_i(o_x) \geq p_i$ and $s_k(o_x) = p_k$ for at least one $1 \leq k \leq n$. For every object $o \neq o_x$ seen during step 1 of our algorithm there is at least one list S_j ($1 \leq j \leq n$) in which o has not been seen and hence we have either $s_j(o) < p_j \leq s_j(o_x)$ *(A)* or $s_j(o) = p_j$ *(B)*.

If case *(A)* applies for an object o, it cannot dominate the object o_x. Thus object o_x can only be dominated by an object for which case *(B)* applies, i.e. one of those objects that have occurred during step 2 of our algorithm. Choose among those objects the maximum object o_m dominating o_x. We will then show by contradiction that o_m belongs to the skyline:

If o_m would not be part of the skyline, it would have to be dominated by another object. Due to Lemma 1 o_m cannot be dominated by any unseen object, and due to being maximal among those objects occurring in step 2, it would have to be dominated by an object o seen in step 1and not seen in step 2 of the algorithm. This means, that there is an index j, such that $s_j(o)$ is smaller than p_j (cf. case 1 above). Therefore o can dominate neither o_x nor o_m leading to the contradiction. ■

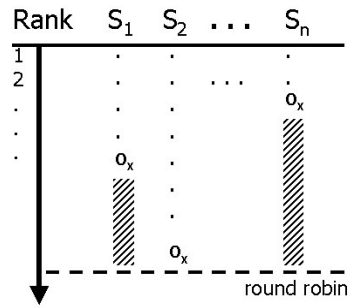

Fig. 3. Savings implemented by heuristic 1

4 Improvements by Advanced Heuristics

Having shown that we will have to see at least one object in all the lists we will now focus on heuristics to find this object that causes the first phase to terminate more quickly and will try to minimize the necessary comparisons within the sets K_i.

Consider the situation shown in figure 3. Having adopted a round robin strategy in our basic algorithm, we have to expand all the lists until an object (e.g. o_x) occurs in all lists. But our proof of correctness allows us to immediately disregard even all those objects that have only occurred in any list *after* o_x (i.e. the shaded areas). Thus using all information about discovered objects at an early stage and employing a sophisticated control flow, we can improve our algorithm by immediately focusing on objects that can be assumed to foster early termination by being the first object to occur in all lists with reasonably high probability. Having chosen such an object we will no longer do sorted accesses on lists in which this object has already occurred, but rather expand lists in which its score is still unknown. Therefore we need to know how to find an object that is most probable to terminate our algorithm. In case studies on multi-objective optimization like [3] one of the most effective functions in estimating dominating objects is a greedy strategy called 'maximin' function. Our heuristic for estimating an appropriate object has been built along the lines of this function. But whereas the 'maximin' function only focuses on the maximum value after evaluating the minimum scores for each object and thus advocates the smallest possible expansion in every list, our heuristic additionally will take advantage of the fact that because of the sorting of the lists and recent sorted accesses on it, we *exactly know* the current score value in each list and thus can better estimate the necessary expansion.

Heuristic 1: If all scores of an object are known (either by sorted or by random access) consider all scores value in lists where it has not yet been seen by sorted access. If we sum up the difference between these values and the last score value seen by sorted access on the respective list, we get an aggregated value for each object. The object with a minimum value can be considered the most promising object. It still needs the least expansion in all lists. Therefore it is probable to be the object that will first occur in all of the lists. If there are more objects with the same minimum score, the one with the minimum sum of scores will need the least expansion of all lists.

To find these objects, we will mix sorted and random accesses already in the first phase to immediately get all information about an object. Since these accesses would have been necessary in the third phase of the basic algorithm anyway, by doing them immediately we just spend a few random accesses too much for those objects that we might already have seen by sorted access in more than one list. Knowing all scores we can now estimate how far we would have to expand all the lists, if we had to see the latest object in all lists adding up the differences between values seen by random access and the current score in the respective list. Focusing only on the best object with respect to necessary list expansions, we will employ indicators that tell us which lists to expand next, avoiding those lists our object has already occurred in.

Since we gather all information about an object at its first occurrence and immediately assess its probable utility for termination, we will not expand any list more than necessary. If we can choose several lists for the next sorted access, we can either pick one randomly, or, if we expect non-uniform data distributions a complementing indicator technique e.g. using the derivatives of the score distribution function in each list along the lines of [9] may be employed to estimate the expected gain in each list. Please note that our heuristic 1 will not affect the abstract order of complexity from our previously stated optimality results, because the maximum improvement factor over the round robin strategy can only be the number of lists (n). But, given the rather expensive costs of object accesses over the Internet even small numbers of accesses saved will improve the overall run-time behavior like shown in [5] or [1]. Thus, also improvements taking only constant factors off the algorithm's complexity should be employed towards meeting real-time constraints.

Our second heuristic will focus on the necessary comparisons within the sets K_i. Obviously no object having a smaller score with respect to S_i will be able to dominate any object having a larger score. Thus we do not really need the pairwise comparisons like suggested in the basic algorithm. We only have to compare pairwise between objects within the same set Ki having equal scores and can otherwise test, if the objects with smaller scores are dominated by ones having larger score values.

Heuristic 2: Start with the objects first seen in each set K_i and compare pairwise all objects with the same score value. Then only test for domination by objects with higher scores.

To implement this we employ the fact that since the lists are ordered, also all K_i are ordered. We will use two counters q and b and divide each K_i into subsets grouping same score values. Starting with the first set we will assume the objects as enumerated and set q to the number of the first object of a subset and b to the number of the last. According to heuristic 2 we don't need any comparisons with objects on numbers larger than b, we need pairwise comparisons for all objects between q and b and we need a test for domination by all objects with numbers smaller than q.

Using our heuristics we will now present our improved algorithm for distributed skyline queries. Again we need the initialization of a central datastructure for the set of possible skyline objects and sets containing the objects for each sorted list as before. Additionally we need a variable for the object that is considered most promising to terminate our algorithm. Note that all necessary random accesses are now already performed in step 1 in order to derive a greedy estimation of the object most probable to foster early termination.

Improved Distributed Skyline Algorithm

0. Initialize a datastructure P := ∅ containing records with an identifier and n real values for scores indexed by the identifiers, initialize n lists $K_1,\ldots,K_n$:= ∅ containing records with an identifier and one real value, initialize a record term_oid containing an identifier and a real value := 0 and initialize n real values $p_1,\ldots,p_n := 1$

1. Initialize counter i := 1.

1.1. Get the next object o_{new} by sorted access on list S_i, set $p_i := s_i(o_{new})$ and update the real value in term_oid according to step 1.3

1.2. If $o_{new} \notin P$

1.2.1. Create a record in P containing oid and score in S_i in the i-th entry in its record.

1.2.2. Do random accesses on all missing scores and update the record in P like above

1.3. Add up the difference between o_{new}'s score values in lists, where it has not yet been seen, and the p_i in these lists

1.4. If this sum is smaller than the value in term_oid, replace the oid and the value in term_oid with the oid and new value of o_{new}

1.5. If the sum is equal to the value in term_oid, replace like in 1.4 only, if the total sum of scores for o_{new} is larger than the sum for the object given by term_oid

1.6. Append o_{new} with $s_i(o_{new})$ to list K_i

1.7. Set i to any number of a set K_i in which the object given by term_oid has not yet occurred. If it is element of all K_i proceed with step 2 else with step 1.1.

2. Let o_{term} be the object given by term_oid. For i = 1 to n do

2.1. While $p_i = s_i(o_{term})$ do sorted access on list S_i and update p_i like in step 1.4, append it to list K_i like in step 1.3 and if the retrieved objects is not in P handle it like in step 1.2.1 and 1.2.2

3. For i = 1 to n do

3.1. q :=0, b:= 0

3.2. While there is an q+1-th entry in K_i do

3.2.1. Repeat (collect an object of K_i and set b:= b+1) until the value of the b+1-th entry in K_i is strictly smaller than the b-th entry or there is no b+1-th entry

3.2.2. If any collected object does not exist in P, discard the object and remove it from K_i and set b := b-1

3.2.3. Compare the collected objects pairwise. If any of these objects is dominated, discard it and remove it from K_i and P and set b := b-1

3.2.4. Compare all collected objects pairwise to all objects being on a position smaller or equal than q in K_i. If any collected object is dominated, discard it and remove it from K_i and P and set b := b-1

3.2.5. Set q := b

4. Output P as the set of all non-dominated objects, i.e. the skyline.

Since the correctness of the improved algorithm is straightforward along the lines of the basic algorithm and also the optimality holds, we will again return to our example to show how the algorithm works. Again we pose a query on the best route with respect to its length (S_1) and the traffic density (S_2). The following tables show our result lists with routes ordered by scores:

S_1 (length)					
R1	R3	R5	R4	R7	…
0.9	0.9	0.8	0.8	0.7	…

S_2 (traffic density)					
R2	R4	R6	R3	R8	…
0.9	0.8	0.8	0.8	0.7	…

The algorithm in step 1 will perform sorted accesses on S_1 and finds route R1. A random access will reveal R1's second score 0.5 and that its sum of unseen values is (1.0-0.5)=0.5. That means our first estimation is that we will have to expand the list S_2 down to score 0.5 in order to see R1 in all lists. Thus we have to do a sorted access on list S_2 trying to decrease the scores to find R1's second score, and we get route R2. The second score of R2 leads to a sum of differences of 0.3. Thus it is more promising than R1 and we will focus on lists where R2 has not yet occurred. Accessing S_1 we encounter object R3, whose second score 0.8 again leads to a change in our term_oid to R3 with value 0.1. After we have also accessed R4 and R6 in list S_2 who both show larger sums, we finally encounter R3 and can terminate step 1.

Route	R1	R2	R3	R4	R6
Score S_1	0.9	0.6	0.9	0.8	0.2
Score S_2	0.5	0.9	0.8	0.8	0.8
term_oid	R1	R2	R3	R3	R3
next access	S_2	S_1	S_2	S_2	S_2

In step 2 we will do some additional accesses on all routes also showing the current minimum score in each list and find that R5 in S_1 already has a smaller score, hence we can discard it, and we can also discard the next object R8 in S_2.

K_1	
R1	R3
0.9	0.9

K_2			
R2	R6	R4	R3
0.9	0.8	0.8	0.8

Step 3 now focuses on the sets K_i and finds that in K_1 R3 dominates R1 and in K_2 we first have to compare R6, R4 and R3 pairwise and find that R3 dominates all and then only have to test, if R3 is dominated by R2. However, as R2 does not dominate R3 we can return them as the skyline. Please note that besides more efficient comparisons within the K_i, even in this limited example our indicator technique already saved us expensive object accesses on routes R5 and R7, which now remain unseen.

5 Evaluation of Distributed Skylining

The presented algorithm for the first time addresses the problem of distributed skylining in Web information systems, thus in our evaluation we obviously cannot compare it to similar algorithms. Since comparisons with algorithms over central indexes (which of course will be faster not having to deal with network latencies) will also yield no sensible results, we will concentrate on the necessary number of object accesses, the total number of objects in the skyline for some practical cases and the

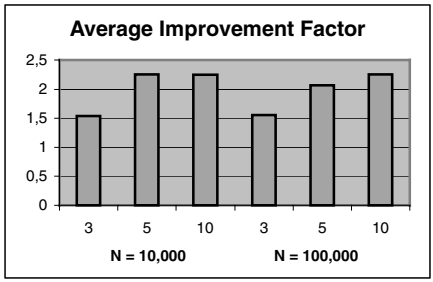

Fig. 4. Improvement due to heuristics

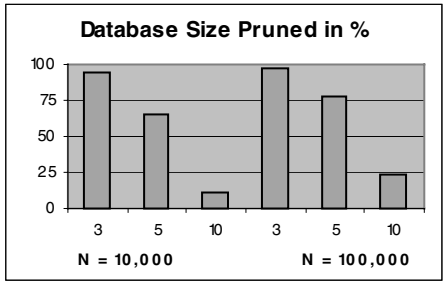

Fig. 5. Saved accesses w.r.t. database size

improvements that can be gained over the basic algorithm by using our advanced heuristics. For all experiments we used an independent data distribution of scores.

Let us first take a glance on the savings due to our heuristics and then evaluate the performance of our improved algorithm. We will focus on the improvement factors in terms of overall object accesses saved. Figure 4 shows the average improvement factors for different numbers of lists (3,5, and 10) and two different database sizes of 10000 and 100000 database objects. We can clearly see that independent of the database size the average improvement factors for our experiments range between 1.5 for small numbers of lists and around 2.5 for higher numbers. Thus, even using just these simple heuristics without any tuning we instantly halve the necessary object accesses. We can even expect higher factors by tuning the given heuristic to adapt more closely to the data distribution like shown e.g. in [9].

Now we can concentrate on the object accesses that our algorithm saves with respect to the database size. Figure 5 shows what percentage of the database can be pruned, again for different numbers of lists and different database sizes. We can see clearly that our algorithm scales well with the database size and for lower numbers of lists works well, e.g. prunes more than 95% over 3 lists. However, we can also see that the performance quickly deteriorates with a growing number of lists. To explain this behavior we have to consider the portions of skyline objects among all objects that have been accessed (cf. figure 6). We find that, though our algorithm's performance seems to deteriorate with growing numbers of lists, its precision in terms of how many objects that are not part of the skyline have to be accessed, heavily increases with growing numbers of lists. For instance in the case of 10 lists over a database of 10000 objects almost 60% of the accesses are definitely necessary to see the entire skyline, i.e. to terminate the algorithm correctly. Considering this instance

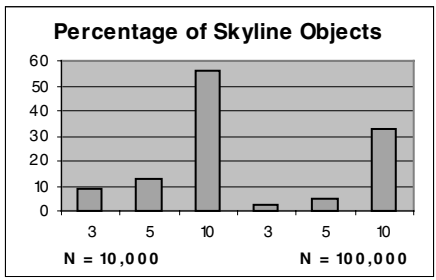

Fig. 6. Skyline objects among all objects accessed

Table 1. Size of the skyline with respect to different numbers of database objects and lists

Size of database (N)	Number of lists	Size of skyline (in % of database size)
10,000	3	0.51
	5	4.44
	10	49.25
100,000	3	0.07
	5	1.00
	10	25.11

further we can conclude that, if we access about 90% of 10000 objects and about 55% of them are necessary, the skyline has to be about 49.5% of the entire database.

To support these considerations we performed more experiments on the actual average size of the skyline for varying numbers of lists and different database sizes. In table 1 we can see that our considerations have been correct (also confirmed by experiments in [4]). Indeed the size of the skyline rapidly increases with larger numbers of lists. We are forced to conclude that, though the concept of skylining may be a very intuitive model for querying, its output behavior seems only to be feasible for rather small numbers of lists to combine. In fact skyline sizes grow exponentially with the number of dimensions. Thus, independently of the retrieval algorithms the problem itself does not scale and we still need a effective dimensionality reduction for skyline queries that are probable to retrieve huge results.

6 Sampling the Efficient Frontier for Improved Scalability

So even if an algorithm could compute high-dimensional skylines in acceptable time, it would still not be sensible to return something like 50% of database objects to the user for manual processing. If on the other hand, users first aggregate all lists in which a compensation between scores can be defined, and then use the skyline query model only for modest numbers of these aggregated lists, the skyline will consist of sensible numbers of elements and can be retrieved reasonably well. But how to know, which dimensions can be compensated and over which dimensions we still need a skyline? As pointed out in [4] specific characteristics of dimensions like correlation have an essential influence on the manageability of the resulting skyline. Correlated data usually results in smaller skylines than the independently distributed case. In contrast anti-correlated distributions amount in a vast increase of the number of skyline objects. Measures to assess such characteristics that hint at the size of the result, are for example the objects' average *consistency of performance*, i.e. if scores for each object show similar absolute values in all different dimensions. The hope is to see *in advance* e.g. if there are correlations between some dimensions, which in turn could be condensed into a single dimension. Since computing skylines of small numbers of dimensions (say 3) are still not at all problematic, our main idea is to get an impression of the original characteristics of the skyline by investigating skylines of some representative low-dimensional subsets of the original dimensions. The following theorem states that -without having to calculate the high-dimensional skyline- our sampling can nevertheless rely on actual skyline objects, which in turn improves the sampling's quality.

Theorem 3 (Skyline of Subsets of Dimensions):
For each object o in the skyline of a subset of the dimensions (i.e. a subset of score lists) there is always a corresponding object o' in the skyline of all dimensions having exactly the same scores as o with respect to the subset of dimensions.

Proof: Assume that we have chosen an arbitrary subset of score lists. We can then calculate the skyline P of this subset. Let o be any object of P. We have to show that there is a corresponding object o' in the skyline Q for all score lists having same scores in the chosen subset. If o already is also part of Q the statement is trivially true. Thus let us assume that o is not element of Q and therefore must be dominated by at least one object p. That means for all lists $s_i(p) \geq s_i(o)$ holds. If, however, considering only the chosen lists there would be any some list for which 'strictly better' holds, i.e. $s_i(p) > s_i(o)$, object o would already be dominated by p with respect to our subset. Since this would be in contradiction to our assumption of o being part of the skyline of the subset, for the entire subset $s_i(p) = s_i(o)$ has to hold and p is our object o'. ■

Using this result we will now propose the sampling scheme. We will sample the skyline in three steps: choosing q subsets of the lists, calculating their lower-dimensional skylines and merging the results as the subsequent sampling. Since skylines can already grow large for only 4 to 5 dimensions, we will always sample with three-dimensional subsets. Values of $q = 5$ for 10 score lists and $q = 15$-20 for 15 score lists in our experiments have provided sufficient sampling quality. For simplicity we just take the entire low-dimensional skyline (2.1)and merge it (2.2). As theorem 3 shows, should two objects feature the same score within a low-dimensional skyline, random accesses on all missing dimensions could be used to rule out a few dominated objects sometimes. We experimented with this (more exact) approach, but found it to perform much worse, while improving the sampling quality only slightly.

Sampling Skylines by Reduced Dimensions

1. Given m score lists randomly select q three-dimensional subsets, such that all lists occur in at least one of the subsets. Initialize the sampling set $P := \emptyset$
2. For each three-dimensional subset do
   - 2.1. Calculate the skyline P_i of the subset
   - 2.2. Union with the sampling set $P := P \cup P_i$
3. The set P is a sample of the skyline for all m score lists

Now we have to investigate the quality of our sampling. An obvious quality measure is the *manageability* of the sample; its number of objects should be far smaller than the actual skyline. Also the *consistency of performance* is also an interesting measure, because larger number of consistent objects will mean some amount of correlation and therefore hints at rather small skylines. Our actual measurement here takes the perpendicular distance between each skyline object and the diagonal in score space $[0,1]^n$ normalized to a value between 0 and 100% and aggregated within 10% intervals. The third measure will be a *cluster analysis* dividing each score dimension into upper and lower half, thus getting 2^m buckets. Our cluster analysis counts the elements in each bucket and groups the buckets according to the number of 'upper halves' (i.e. score values > 0.5) they contain. Again having more elements in the clusters with either higher or lower numbers of 'upper halves' indicate correlation, whereas objects in the buckets with medium numbers hint at anti-correlation.

Our experiments on how adequately the proposed sampling technique predicts the actual skyline, will focus on a 10-dimensional skyline for a database containing

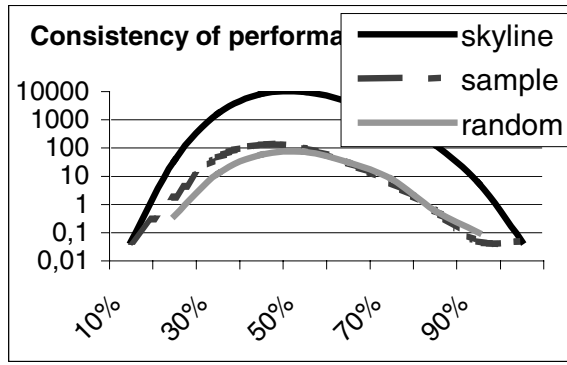

Fig. 7. Consistency of performance

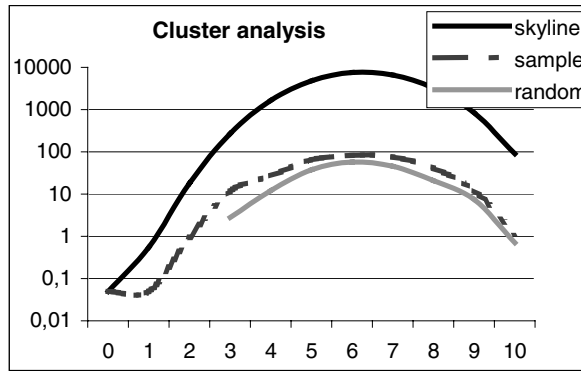

Fig. 8. Cluster analysis for the 10-dim sample

N=100,000 objects. Score values over all dimensions have been uniformly distributed and statistical averages over multiple runs and distributions have been taken. We have fixed $q := 5$ and compare our measurement against the quality of a random sample of the actual 10-dimensional skyline, i.e. the best sample possible (which, however, in contrast to our sample cannot be taken without inefficiently calculating the high-dimensional skyline). Since our sample is expected to essentially reduce the number of objects, we will use a logarithmic axis for the numbers of objects in all diagrams.

We have randomly taken 5 grips of 3 score lists and processed their respective skylines like shown in our algorithm. Measuring the manageability we have to compare the average size of the 10-dim skyline and our final sample: the actual size of the skyline is on average 25133.3 objects whereas our sample consists of only 313.4 objects, i.e. 1.25% of the original size. Figure 7 shows the consistency of performance measure for the actual skyline, our sample and a random sample of about the same size as our sample. The shapes of the graphs are quite accurate, but whereas the peaks of the actual set (dark line) and its random sample (light line) are aligned, the peak for our sampling (dashed line) is slightly shifted to the left. We thus underestimate the consistency of performance a little, because when focusing on only

a subset of dimensions, some quite consistent objects may 'hide' behind optimal objects with respect to these dimensions, having only slightly smaller scores, but nevertheless a better consistency. But this effect only can lead to a slight overestimation of the skyline's size and thus is in tune with our intentions of preventing the retrieval of huge skylines. Figure 8 addresses our cluster analysis. Again we can see that our sampling graph snugly aligns with the correct random sampling and the actual skyline graph. Only for the buckets of count 3 there is a slight irritation, which is due to the fact that we have sampled using three dimensions and thus have definitely seen all optimal objects with scores >0.5 in these three dimensions. Thus we slightly overestimate their total count. Overall we see that our sampling strategy with reduced dimensions promises -without having to calculate the entire skyline- to give us an impression of the number of elements of the skyline almost as accurate as a random sample of the actual skyline would provide. Using this information for either safely executing queries or passing them back to the user for reconsideration in the case of too many estimated skyline objects seems promising to lead to a better understanding and manageability of skyline queries.

7 Summary and Outlook

We addressed the important problem of skyline queries in Web information systems. Skylining extends the expressiveness of the conventional 'exact match' or the 'top k' retrieval models by the notion of Pareto optimality. Thus it is crucial for intuitive querying in the growing number of Internet-based applications. Distributed Web Information services like [5] or [2] are premium examples benefiting from our contributions. In contrast to traditional skylining, we presented a first algorithm that allows to retrieve the skyline over distributed data sources with basic middleware access techniques and have proven that it features an optimal complexity in terms of object accesses. We also presented a number of advanced heuristics further improve performance towards real-time applications. Especially in the area of mobile information services [22] using information from various content providers that is assembled on the fly for subsequent use, our algorithm will allow for more expressive queries by enabling users to specify even complex preferences in an intuitive way. Confirming our optimality results our performance evaluation shows that our algorithm scales with growing database sizes and already performs well for reasonable numbers of lists to combine. To overcome the deterioration for higher numbers of lists (curse of dimensionality) we also proposed an efficient sampling technique enabling us to estimate the size of a skyline by assessing the degree of data correlation. This sampling can be performed efficiently without computing high-dimensional skylines and its quality is comparable to a correct random sample of the actual skyline.

Our future work will focus on the generalization of skylining and numerical top k retrieval towards the problem of multi-objective optimization in information systems, e.g. over multiple scoring functions like in [10]. Besides we will focus more closely on quality aspects of skyline queries. In this context especially a-posteriori quality assessments along the lines of our sampling technique and qualitative assessments like in [17] may help users to cope with large result sets. We will also investigate our proposed quality measures in more detail and evaluate their individual usefulness.

Acknowledgements. We are grateful to Werner Kießling, Mike Franklin and to the German Research Foundation (DFG), whose Emmy-Noether-program funded part of this research.

References

1. W.-T. Balke, U. Güntzer, W. Kießling. On Real-time Top k Querying for Mobile Services, In *Proc. of the Int. Conf. on Coop. Information Systems (CoopIS'02)*, Irvine, USA, 2002
2. W.-T. Balke, W. Kießling, C. Unbehend. A situation-aware mobile traffic information prototype. In *Hawaii Int. Conf. on System Sciences (HICSS-36)*, Big Island, Hawaii, USA, 2003
3. R. Balling. The Maximin Fitness Function: Multi-objective City and Regional Planning. In *Conf. on Evol. Multi-Criterion Optimization (EMO'03),* LNCS 2632, Faro, Portugal, 2003
4. S. Börzsönyi, D. Kossmann, K. Stocker. The Skyline Operator. In *Proc. of the Int. Conf. on Data Engineering (ICDE'01)*, Heidelberg, Germany, 2001
5. N. Bruno, L. Gravano, A. Marian. Evaluating Top-k Queries over Web-Accessible Databases. In *Proc. of the Int. Conf. on Data Engineering (ICDE'02)*, San Jose, USA, 2002.
6. J. Chomicki. Querying with intrinsic preferences. In *Proc. of the Int. Conf. on Advances in Database Technology (EDBT)*, Prague, Czech Republic, 2002
7. R. Fagin, A. Lotem, M. Naor. Optimal Aggregation Algorithms for Middleware. *ACM Symp. on Principles of Database Systems (PODS'01)*, Santa Barbara, USA, 2001
8. P. Fishburn. *Preference Structures and their Numerical Representations*. Theoretical Computer Science, 217:359-383, 1999
9. U. Güntzer, W.-T. Balke, W. Kießling. Optimizing Multi-Feature Queries for Image Databases. In *Proc. of the Int. Conf. on Very Large Databases (VLDB'00)*, Cairo, Egypt, 2000
10. R. Keeney, H. Raiffa. Decisions with Multiple Objectives: Preferences and Value Tradeoffs. Wiley & Sons, 1976
11. W. Kießling. Foundations of Preferences in Database Systems. In *Proc. of the Int. Conf. on Very Large Databases (VLDB'02)*, Hong Kong, China, 2002
12. W. Kießling, G. Köstler. Preference SQL - Design, Implementation, Experiences. In *Proc. of the Int. Conf. on Very Large Databases (VLDB'02)*, Hong Kong, China, 2002
13. D. Kossmann, F. Ramsak, S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Conf. on Very Large Data Bases (VLDB'02)*, Hong Kong, China, 2002
14. H. Kung, F. Luccio, F. Preparata. On Finding the Maxima of a Set of Vectors. *Journal of the ACM*, vol. 22(4), ACM, 1975
15. M. Lacroix, P.Lavency. Preferences: Putting more Knowledge into Queries. In *Proc. of the Int. Conf. on Very Large Databases (VLDB'87)*, Brighton, UK, 1987
16. Map-Quest Roadtrip Planner. www.map-quest.com, 2003
17. M. McGeachie, J. Doyle. Efficient Utility Functions for Ceteris Paribus Preferences. In *Conf. on AI and Innovative Applications of AI (AAAI/IAAI'02)*, Edmonton, Canada, 2002
18. NTT DoCoMo home page. http://www.nttdocomo.com/home.html, 2003
19. M. Ortega, Y. Rui, K. Chakrabarti, et al. Supporting ranked boolean similarity queries in MARS. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, Vol. 10 (6), 1998
20. D. Papadias, Y. Tao, G. Fu, et.al. *An Optimal and Progressive Algorithm for Skyline Queries.* In *Proc. of the Int. ACM SIGMOD Conf. (SIGMOD'03),* San Diego, USA, 2003
21. K.-L. Tan, P.-K. Eng, B. C. Ooi. *Efficient Progressive Skyline Computation*. In Proc. *of Conf. on Very Large Data Bases (VLDB'01)*, Rome, Italy, 2001
22. M. Wagner, W.-T. Balke, et al.. A Roadmap to Advanced Personalization of Mobile Services. In *Proc. of the. DOA/ODBASE/CoopIS (Industry Program), Irvine,* USA, 2002.