Parallel Distributed Processing of Constrained Skyline Queries by Filtering

Bin Cui [1], Hua Lu [2], Quanqing Xu [1], Lijiang Chen [1], Yafei Dai [1], Yongluan Zhou [3]

[1]*Department of Computer Science, Peking University, China*
{bin.cui, xqq, clj, dyf}@pku.edu.cn

[2]*Department of Computer Science, Aalborg University, Denmark*
luhua@cs.aau.dk

[3] *Distributed Information Systems Laboratory, EPFL, Switzerland*
yongluan.zhou@epfl.ch

***Abstract*— Skyline queries are capable of retrieving interesting points from a large data set according to multiple criteria. Most work on skyline queries so far has assumed a centralized storage, whereas in practice relevant data are often distributed among geographically scattered sites. In this work, we tackle constrained skyline queries in large-scale distributed environments without the assumption of any overlay structures, and propose a novel algorithm named $PaDSkyline$ (Parallel Distributed $Skyline$ query processing). $PaDSkyline$ significantly shortens the response time by performing parallel processing over site groups produced by a partition algorithm. Within each group, it locally optimizes the query processing over distributed sites. It also drastically enhances the network transmission efficiency by performing early reduction of skyline candidates with deliberately selected multiple filtering points. Results of extensive experiments demonstrate the efficiency and robustness of our proposals.**

I. Introduction

A skyline query [1] returns *interesting* points from a set of multiple dimensional data points. A point is said to be interesting if it is not *dominated* by any other points. A point pt_1 is said to dominate pt_2, if pt_1 is not worse than pt_2 in every single dimension but better than pt_2 in at least one dimension. The meaning of "better" varies in different situations, for example "smaller" or "larger" in value comparison, and "earlier" or "later" in date comparison. Because of their powerful capability of retrieving interesting points from a large n-dimensional data set, skyline queries are well suitable for applications like decision making and multiple criteria optimization. For instance, a tourist can issue a skyline query on a hotel relation to get those ones with good recommendations and cheap prices.

Most work on skyline queries [1], [2], [3], [4], [5], [6] so far has assumed a centralized data storage, and been focused on providing efficient skyline computation algorithms on a sole database. This assumption, however, fails to reflect the distributed computing environments consisting of different computers, which are located at geographically scattered sites and connected via Internet. For example, a stock trader needs to know which stocks worldwide are worth investing, based on the trading records of the previous day. For this purpose, he has to access multiple stock information databases available at different places like New York Stock Exchange, London Stock Exchange, Tokyo Stock Exchange, etc. For each single stock, the agent needs to take into consideration multiple attributes like *last trade price*, *change*, *last close price*, *estimated price*, *volume*, etc. Therefore, a skyline query against those distributed databases will help the agent get those interesting stocks.

Another real-life example is online comparative shopping, in which a search engine needs to get good bargains from many different shopping sites according to multiple criteria like *price*, *quality*, *guarantee*, etc. Clearly, such multiple criteria are best captured by a skyline query.

In such cases as mentioned above, however, directly applying existing techniques to process skyline queries would incur large overhead. In this work, we intend to efficiently process constrained skyline queries [5] in such widely distributed environments. We propose a novel algorithm named *PaDskyline* which stands for *Pa*rallel *D*istributed *Skyline* query processing. A constrained skyline query is attached with constraints on specific dimensions. A constraint on a dimension is a range specifying the user's interest. Refer again to the stock selection example. The agent may only be interested in those stocks whose *last trade prices* are between \$15 to \$20. Similar constraints are also applicable to other dimensions. Two points are noteworthy about possible constraints. One is that the range denoted by a constraint can be unclosed, like *estimated price* higher than \$20. The other is that constraints can be available in only part of the total n dimensions.

Given a distributed environment as aforementioned, our objective is efficient query processing strategies that shorten the overall query response time. We first speed up the overall query processing by achieving parallelism of distributed query execution. Given a skyline query with constraints, all relevant sites are partitioned into incomparable groups among which the query can be executed in parallel. The parallel execution also makes it possible to report skyline points progressively, which is usually desirable to users. Within each group, specific plans are proposed to further improve the query processing

978-1-4244-1837-4/08/\$25.00 © 2008 IEEE

involving all intra-group sites. On a processing site, multiple filtering points are deliberately picked based on their overall dominating potential from the local skyline. They are then sent to other sites with the query request, where they help identify more unqualified points otherwise reported as false positives, and thus reducing the communication cost between data sites.

We make the following contributions on the problem of distributed constrained skyline queries in a network environment without assuming any overlay structures. First, we propose a specific partition algorithm that divides all relevant sites into groups, such that a given query can be executed in parallel among all those site groups. Second, we elaborate the query execution within any single group of sites, which includes deciding a query forwarding plan between sites and designating a group head responsible for query results merger. Third, we give a parallel distributed skyline algorithm based on the first two contributions, together with a cost model to estimate query costs. Fourth, we detail how to use multiple filtering points in distributed query processing such that the network data transmission is more efficient. Two simple yet efficient heuristics are developed for selecting multiple filtering points on a processing site. Finally, we conduct extensive experiments on both synthetic and real datasets, and the results demonstrate the efficiency and robustness of our proposals.

The remainder of this paper is organized as follows. Section II provides a brief review of related work on skyline queries. Section III gives the problem definition. In Sections IV and V we detail the parallel distributed query execution and the selection of multiple filtering points, respectively. Section VI presents the experimental results. Finally, we conclude the paper in Section VII.

II. Related Work

Borzonyi et al. [1] introduced the skyline operator into database systems with algorithms *Block Nested Loop* (BNL) and *Divide-and-Conquer* (D&C). Chomicki et al. [2] proposed a *Sort-Filter-Skyline* (SFS) algorithm as a variant of BNL. Tan et al. [6] proposed two progressive algorithms: *Bitmap* and *Index*. The former represents points in bit vectors and employs bit-wise operations, while the latter utilizes data transformation and B^+-tree indexing. Kossmann et al. [4] proposed a *Nearest Neighbor* (NN) method. It identifies skyline points by recursively invoking R*-tree based depth-first NN search over different data portions. Papadias et al. [5] proposed a *Branch-and-Bound Skyline* (BBS) method based on the best-first nearest neighbor algorithm [7]. Godfrey et al. [3] provided a comprehensive analysis of previous skyline algorithms without indexing supports, and proposed a new hybrid method with improvement. All these works assume centralized data storage.

Deviating from skyline queries in the centralized setting, Balke et al. [8] addressed skyline operation over web databases where different dimensions are stored in different data sites. Their algorithm first retrieves values in every dimension from remote data sites using sorted access in round-robin on all dimensions. This continues until all dimension values of an object, called the terminating object, have been retrieved. Then all non-skyline objects will be filtered from all those objects with at least one dimension value retrieved. Differently in this paper, our work deals with distribution of data horizontally partitioned.

Wu et al. [9] proposed a parallel execution of constrained skyline queries in a CAN [10] based distributed environment. By using the query range to recursively partition the data region on every data site involved, and encoding each involved (sub-)region dynamically, their method avoids accessing sites not containing potential skyline points and progressively reports correct skyline points. Wang et al. [11] developed Skyline Space Partitioning (SSP) approach to compute skylines on a tree-structured P2P platform BATON. SSP partitions the skyline space into regions and maps them in a single dimensional order, which allows regions to be distributed to different peer nodes according to BATON protocols. Our proposal in this paper differs from these two pieces of work in that we do not assume any overlay availability on top of the original network.

Huang et al. [12] proposed techniques for skyline query processing in MANETs. Lightweight devices in MANETs are able to issue spatially constrained skyline queries that involve data stored on many mobile devices. Queries are forwarded through the whole MANET without routing information. They proposed a filtering based data reduction technique that reduces the data transferred among devices. Our work, assuming a wired large-scale distributed environment, is also different from this work in a MANET setting.

III. Problem Definition

Given a set of m sites $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ distributed at different geographic locations, each S_i has a local relation R_i. Every tuple in any R_i is n-dimensional point, represented as $(p_1, p_2, \ldots, p_n)$. Different R_is may overlap, i.e., it is possible that $R_i \cap R_j \neq \emptyset$ for $i \neq j$.

Without loss of generality, we assume smaller values are preferred in the skyline operator. We use $pt_1 \prec pt_2$ to represent point pt_1 dominates point pt_2. Besides, we suppose any site S_{org}, able to directly communicate with any other site $S_i \in \mathcal{S}$ through wired end-to-end connections, may initiate against all R_is a skyline query with a set of n constraints $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$. Each C_i is either a range $[l_i, u_i]$, or a $\emptyset$ indicating no constraint in that dimension. Our goal is to get the result for the constrained skyline query efficiently, i.e., with short response time. We define the query response time as the time period from the moment a query is issued by a site S_{org}, to the moment S_{org} receives the complete and correct answers after contacting other sites.

To shorten the response time of a query, we mainly endeavor on two aspects. On one hand, we propose effective ways to guide deciding the query forwarding and execution order between different sites, so as to obtain the query execution parallelism, and boost the result reporting progressiveness. On the other hand, we generalize the single filtering point idea [12] to use multiple filtering points, and thus enhancing the filtering power and reducing the amount of data transmitted between remote sites. We propose benefit measurements and

heuristics to guide the selection of powerful multiple filtering points.

In contrast to previous work [11], [9], our problem definition does not assume the availability of an overlay network where different nodes hold disjoint data partitions. Instead, different relations on different sites may overlap. Therefore, these previous methods are not applicable to our problem.

IV. Parallel Distributed Skyline Processing

A. Motivation

In the previous work [12], a skyline query is forwarded among mobile peers via multiple hops according to the wireless nature of MANETs [13]. Whereas in a wired environment, connections are end-to-end. Because of such wired connections between each pair of sites, a constrained skyline query can be sent out to all peer sites and then each site can execute the query on its own data set simultaneously. This naive execution plan might benefit from the parallelism among all peer sites. However, it is very sensitive to large local skyline results, usually of but not limited to anti-correlated datasets [1], which lead to heavy communication cost. For this reason, identifying unqualified candidates in local skylines early is favorable.

Following the data reduction principle in semijoin in distributed query processing [14], [15], Huang et al. [12] proposed to transfer a single local skyline point with the query request among mobile peers, which acts as a filter to identify those unqualified ones in peers' local skylines. In this work, we extend this single filtering point to multiple ones since wired connections are much faster and more reliable than wireless channels. How to choose multiple filtering points will be detailed in Section V. In this section, we focus on finding a good distributed skyline query execution plan that minimizes the total execution time. We compare the naive approach with the parallel distributed approach with multiple filtering points we will propose, using a cost model taking into account both local processing time and network transmission time.

We argue that the order to execute a distributed skyline query among multiple sites really matters, because an appropriate execution order can filter more data points earlier and thus reducing not only communication cost but also the local processing cost in the subsequent execution. We intend to balance the parallelism and filtering when processing a distributed skyline query (DSQ for short) and get short overall response time.

B. Parallel Distributed Query Execution

To help determine execution order among different sites, the query originator first asks each site S_i for its MBR_i, the n-dimensional minimum bounding box of the local relation R_i. If a site's MBR disjoins with the constraint set $\mathcal{C}$ specified in the query, it will not be considered in the following query processing. For each site whose MBR_i overlaps with $\mathcal{C}$, we only need to consider the intersection $MBR_i \cap \mathcal{C}$. We call this intersection *reduced minimum bounding box* and use $rMBR_i$ to represent it.

We proceed to partition all those sites left into several groups according to their $rMBR$s, such that the skyline computation in any one group does not depend on or affect the computation in any other group. Therefore, the given skyline query can be executed in parallel among those site groups. While within any individual site group the query is executed according to some local plans, which will be discussed in Section IV-B.2.

1) Incomparable Partition of Sites and Parallelism: Given two data sites S_i and S_j (with their reduced minimum bounding boxes $rMBR_i$ and $rMBR_j$ respectively), we need to determine if the skyline query can be executed against them in a parallel way such that two results do not affect each other. Intuitively, we cannot do this if they overlap, as in the overlapping region points from different boxes may be not incomparable. Whereas, the non-overlapping relationship does not necessarily permit parallelism. To help deal with this problem, we first give a definition of "incomparable" between two data sites.

Definition 1: Two data sites S_i and S_j are incomparable with respect to the constrained skyline query, iff $rMBR_i.min.DR \cap rMBR_j = \emptyset$ and $rMBR_j.min.DR \cap rMBR_i = \emptyset$.

In Definition 1 above, $rMBR_i.min.DR$ stands for the dominating region of $rMBR_i$'s minimum corner with respect to the constraints specified in the skyline query. It is obtained as the intersection of the original dominating region [12] and the set of constraints $\mathcal{C}$, which will be formalized in Section V-A.1. When there is no confusion or ambiguity in the context, we also say that two reduced minimum bounding boxes $rMBR_i$ and $rMBR_j$ are incomparable, which actually means that their corresponding data sites are incomparable. With this definition, we have the following lemma.

Lemma 1: If two data sites S_i and S_j are incomparable with respect to the constrained skyline query, it holds that $\forall pt_i \in rMBR_i$, $\nexists pt_j \in rMBR_j$ s.t. $pt_i \prec pt_j$ and $\forall pt_j \in rMBR_j$, $\nexists pt_i \in rMBR_i$ s.t. $pt_j \prec pt_i$.

The correctness of this lemma is guaranteed by the property that the minimum corner of $rMBR$ has the strongest dominating capability among all possible points in $rMBR$. This lemma shows that a given skyline query can be executed against two incomparable sites in a parallel way without conflict between two results. It is easy to see that this parallelism can be generalized to multiple sites, any pair of which are incomparable. A by-product of this parallelism is the progressiveness of skyline reporting, because the result of any single site among those pair-wise incomparable sites definitely appears in the final skyline.

It is not impossible that, however, all sites are not pair-wise incomparable, as we do not assume any partition available among all sites. Thus, we partition the set of sites $\mathcal{S}$ into non-empty groups that satisfy: (1) Any pair of sites from different groups are incomparable; (2) For any pair of sites S_i and S_j within the same non-singleton group, there exists a path $S_i, S_k, \ldots, S_j$ such that any adjacent pair along the path are not incomparable. Property 1 ensures that parallel execution

can be carried out against groups of sites. While property 2 clusters together those sites whose local skyline results potentially conflict, and provides opportunity for alternative optimizations. We call the partition *incomparable partition* of sites.

The partition algorithm, $icmpPartition$ for short, is shown in Figure 1. Before the real partitioning, intersections of site MBRs and constraint set $\mathcal{C}$ are obtained with unpromising sites being pruned (lines 1–3). The partitioning procedure starts by assigning the first candidate site into a singleton group (line 4). Then, each remaining site S_i is compared with every current group, and any group containing sites not incomparable with S_i is removed from the partition into a temporary group $\overline{\mathcal{S}_i}$ (line 7-9). At the end of each loop S_i is assigned into the adjusted partition, either in a new singleton group or in the temporary group with other relevant sites found in the loop (line 10).

Algorithm icmpPartition($\mathcal{S}$, $\mathcal{C}$)
Input: $\mathcal{S}$ is the set of data sites
$\mathcal{C}$ is the set of constraints in the skyline query
Output: an incomparable partition of $\mathcal{S}$
// Adjust MBRs and prune unqualified sites
1. **for each** $S_i \in \mathcal{S}$
2. $rMBR_i = MBR_i \cap \mathcal{C}$;
3. **if** ($rMBR_i == \emptyset$) $\mathcal{S} = \mathcal{S} - \{S_i\}$;
// Compute the independent partition of all relevant sites
4. $\Pi_{\mathcal{S}} = \{\{S_{1'}\}\}$; // $S_{1'}$ is the current 1^{st} element in $\mathcal{S}$
5. **for each** $S_i \in \mathcal{S} - \{S_{1'}\}$
6. $\overline{\mathcal{S}_i} = \emptyset$;
7. **for each** $\mathcal{S}_i \in \Pi_{\mathcal{S}}$
8. **if** ($\exists S_j \in \mathcal{S}_i$ s.t. S_j and S_i are not incomparable)
9. $\Pi_{\mathcal{S}} = \Pi_{\mathcal{S}} - \{\mathcal{S}_i\}$; $\overline{\mathcal{S}_i} = \overline{\mathcal{S}_i} \cup \mathcal{S}_i$;
10. $\Pi_{\mathcal{S}} = \Pi_{\mathcal{S}} \cup \{\{S_i\} \cup \overline{\mathcal{S}_i}\}$

Fig. 1. Incomparable Partition Algorithm

Refer to an example in 2-dimensional space shown in Figure 2, where $\mathcal{S} = \{A, B, C, D, E, F, G\}$. Each site's MBR is shown in a corresponding rectangles. Based on the incomparableness related properties above, $\mathcal{S}$ is partitioned into $\{\{A\}, \{B, C, D, E\}, \{F, G\}\}$. A forms a singleton group because it is incomparable with any other site. Though B and D are incomparable, they are assigned to the same group with C and E, because either of them are not incomparable with C (and E). F and G are incomparable with any other site but not with each other, therefore they constitute another group.

2) Intra-Group Query Execution: Within each group of the sites partition obtained above, we need to consider the execution order among those relevant sites. Now effective filtering is our concern, in other words we use filtering points when forwarding a DSQ between sites in the same group. Given a group $\mathcal{S}_i = \{S_{i_1}, \ldots, S_{i_k}\}$, we have two alternatives for intra-group execution order.

A simple way is to decide a sequence of these sites and

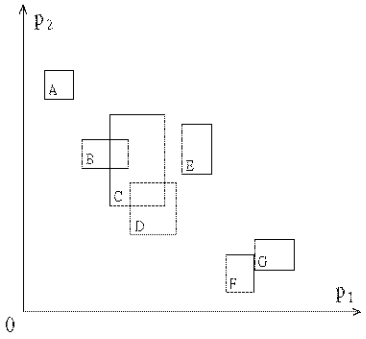


Fig. 2. Incomparable Partition of Sites

forward the DSQ with filtering points along the sequence. The sequence is gained by sorting all sites S_{i_j}s in a non-descending order of the Euclidean distance between the minimum corners of $rMBR_{i_j}$ and constraints set $\mathcal{C}$. The reason for this lies in the intuition that a point nearer to the minimum corners of $\mathcal{C}$ (or the origin when a skyline query without constraints is concerned) is more powerful in terms of dominating capability. The sorting can be integrated into the partition algorithm in Figure 1, on line 9 (line 10) where $\overline{\mathcal{S}_i}$ and $\mathcal{S}_i$ ($\{S_i\}$) are united.

The linear approach ignores any internal possibilities for parallelism. In Figure 2, for example, the query can be parallelized against sites B and D either before or after other sites. To fully capture this potential for intra-group parallelism, a directed graph is needed to store all sites in a group, which is quite complicated due to the nature of a graph. As a trade-off between complexity and parallelism, we use a tree to organize each group of sites. For this purpose, we adapt the partition algorithm in Figure 1 as follows. In each loop (lines 5–10) on site S_i, instead of using a temporary group $\overline{\mathcal{S}_i}$ to contain the union of all groups removed out from $\Pi_{\mathcal{S}}$, S_i is used as a root and all such groups are attached to it as children, and finally the tree is added into the partition as an updated group. After the partitioning, within every group the query is executed and forwarded with filtering points in the top-down manner starting from the root.

For either way, every group needs an assembly to merge results from sites and remove false positives during the query processing. We install this procedure in the *group head* which, responsible for returning result to query originator, is the first site in the linear approach or the root in the tree-base approach.

C. Parallel Distributed Skyline Algorithm

Based on the discussion so far, our overall parallel distributed skyline algorithm, called PaDSkyline, is presented in Figure 3. When a site S_{org} issues a distributed skyline query with constraints $\mathcal{C}$, it first gets the incomparable partition of all sites by calling icmpPartition (line 1). After that, the query request will be sent out to each group head in parallel (lines 2–3). The query request includes constraints $\mathcal{C}$, network addressable query originator identifier S_{org}, and the intra-group query execution plan $g_i.plan$. Next, S_{org} enters a wait, during which it will be triggered by an incoming result reply

from a group head, until all of them have replied (lines 4–7). Upon receiving the reply from the head of group g_i, S_{org} directly reports the received result $g_i.result$ (lines 5–6) as it is incomparable with results from other groups.

Algorithm PaDSkyline($\mathcal{S}$, $\mathcal{C}$)
Input: $\mathcal{S}$ is the set of data sites
$\mathcal{C}$ is the set of constraints in the skyline query
Output: the constrained skyline
// get the incomparable partition of all sites
1. $\Pi_{\mathcal{S}} = \text{icmpPartition}(\mathcal{S}, \mathcal{C})$
// parallel execution
2. **for each** group $g_i \in \Pi_{\mathcal{S}}$ in parallel
3. send $\langle \mathcal{C}, S_{org}, g_i.plan \rangle$ to g_i's group head
// result merge, triggered by incoming result reply
4. **repeat**
5. receive result reply from a group g_i's head
6. report $g_i.result$
7. **until** all group heads have replied

Fig. 3. Parallel Distributed Algorithm

For the sake of simplicity, we designate S_{org} as the head of its site group. For that case, the network communication between S_{org} and a group head degrades to an inter-process or inter-thread communication on a single host.

After a group g_i's head receives the query request, it carries out the intra-group skyline computation as shown in Figure 4. First, it computes its local constrained skyline R_g and captures the initial filtering points set $F_{\mathcal{C}}$ during the local computation (line 1). How to select multiple filtering points will be detailed in Section V. Next, it sends the query request further to downstream site(s) in the group query plan (line 2). Note here the query request includes the constraints $\mathcal{C}$, network addressable group head's identifier S_g, the reduced query plan $plan'$, and the initial filtering points set $F_{\mathcal{C}}$. Each site S_i receiving the query request computes its local result $S_i.R$, returns $S_i.R$ to site S_g, and sends the query request with updated filtering points and further reduced plan to its own downstream site(s). While S_g keeps receiving $S_i.R$ and merging it with R_g until all group members have replied (lines 3–6).

During query execution within a site group, the filtering points set $F_{\mathcal{C}}$ will be dynamically changed, as will be covered in Section V-C. Also, the query plan is dynamically reduced as follows. Each site, including the group head, removes itself from the plan. If the plan is a single sequence, the reduced plan is the sub-sequence left and its head is exactly the target downstream site. If the plan is a tree, the removal may result in more than one sub-trees. For that case, each of them will be sent to a corresponding target downstream site, which is exactly the root of a sub-tree.

D. Cost Model

Suppose for a given local relation R_i, the time to compute its local skyline SK_i is $T_l(R_i)$. We use "$|SK_i|$" to represent the size in bytes of a local skyline. The time to transfer SK_i between two sites is determined by its size, together with the network relevant conditions like bandwidth.

Algorithm groupSkyline($\mathcal{C}, S_{org}, plan$)
Input: $\mathcal{C}$ is the set of constraints in the skyline query
S_{org} is the query originator site identifier
$plan$ is the query execution plan in the group
Output: the constrained skyline within the group
1. compute local skyline R_g,
and get the initial filtering points set $F_{\mathcal{C}}$
2. send $\langle \mathcal{C}, S_g, plan', F_{\mathcal{C}} \rangle$ to next site(s) in $plan$
// result merge, triggered by incoming result reply
3. **repeat**
4. receive result reply from a group member S_i
5. merge $S_i.R_i$ with R_g,
removing duplicates and false positives
6. **until** all group members have replied
7. **return** R_g to S_{org}

Fig. 4. Intra-Group Algorithm

In the naive approach without filtering (see Section IV-A), the time for query originator S_{org} to get result from a peer site S_i is the sum of the local computation time and network transmission time, as shown in the following Formula 1.

$$T_i = T_l(R_i) + T_{tran_{i,org}}(|SK_i|) \tag{1}$$

Due to the parallelism of a naive query execution, its overall response time T_{nav} is determined by the peer site whose result reaches S_{org} latest and the local assembly time on S_{org}, as shown in the following Formula 2.

$$T_{nav} = max\{T_i \mid 1 \le i \le m\} + T_{asm} \tag{2}$$

Suppose an incomparable partition $\Pi_{\mathcal{S}} = \{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ is produced for a given distributed skyline query with constraints, then its overall response time T_{pdq} is determined by the group whose result reaches S_{org} latest, as shown in Formula 3. Note in the parallel distributed query processing, we do not need a global assembly on S_{org}.

$$T_{pdq} = max\{T_{\mathcal{S}_i} \mid 1 \le i \le k\} \tag{3}$$

In the formula above $T_{\mathcal{S}_i}$ is the response time of group $\mathcal{S}_i$, i.e., the time lapse before S_{org} receives the result from $\mathcal{S}_i$'s group head h_i. It is detailed in the following Formula 4. T_{h_i} is the local response time, i.e., the time lapse before h_i gets all results from sites in the group. $T_{asm_{h_i}}$ is the local assembly time, and $SK_{\mathcal{S}_i}$ is the result sent to S_{org}.

$$T_{\mathcal{S}_i} = T_{h_i} + T_{asm_{h_i}} + T_{tran_{h_i,org}}(|SK_{\mathcal{S}_i}|) \tag{4}$$

E. Pre-Computation of Local Skylines

If a skyline query is not attached with constraints, the local skyline result SK_i on any single site S_i can be directly used to compare with filtering points. Similarly, if a site S_i's data set is fully contained within the constraints $\{C_1, C_2, \ldots, C_n\}$

specified in a skyline query, the local skyline SK_i is still directly useful. With this observation, we pre-compute for each site S_i its local skyline SK_i without any constraint, and store SK_i to expect that it can be reused by future queries. With this local optimization we hope to reduce the local skyline query processing times, and thus shortening the overall query response times.

V. Enhancement of Filtering Points

In [12], one single filtering point is transferred and changed from peer to peer. During the query forwarding and processing, the single filtering point is dynamically changed once another point is found to be more powerful in filtering out unqualified candidates.

By contrast, as the network connections in this work are wired with considerably steady and high bandwidth compared to wireless MANETs, we turn to use multiple filtering points among sites, expecting to get higher filtering power. Then, we need to decide which skyline points should be used as filtering points to achieve the best data reduction rate in the distributed skyline query processing.

In this section, we first formalize the dominating region of multiple skyline points with respect to constraints, next we address how to select multiple filtering points initially, and then we proceed to discuss how to dynamically change filtering points during the query processing.

A. Dominating Region of K Filtering Points

Filtering points are selected from the local skyline result initially obtained. Suppose the initial skyline result is $SK_{init} = \{s_1, s_2, \ldots, s_l\}$, we need to select K ($< l$) points from it as the multiple filtering points. The concrete value of K is constrained by l and has effect on network communication cost and pruning capacity obtained. It is not possible to give a generic optimal value of K for all situations. We instead experimentally study this issue in Section VI-B.

1) Dominating Region with Constraints: When K equals 1, we select the point with the maximum potential of dominating others, which is measured by the volume of a point's *dominating region*, the hyper-rectangle determined by the point and the maximum corner of the data space [12]. Figure 5 shows a 2-dimensional example.

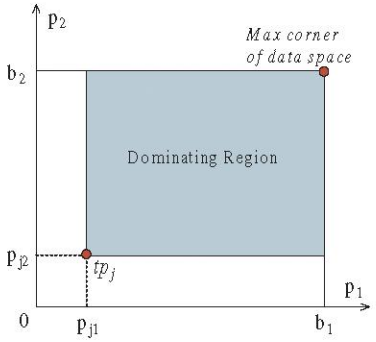


Fig. 5. Dominating Region without Constraints

In the presence of the constraints specified for relevant dimensions, the definition of dominating region needs amendments accordingly. As shown in Figure 6, suppose a constraint range $[l_1, u_1]$ is specified on the dimension p_1. Consequently, for a point tp_j ($<p_{j1}, \ldots, p_{jn}>$), its dominating region shrinks to the smaller one determined by its own value, constraint upper bound u_1, and dimension p_2's upper bound b_2.

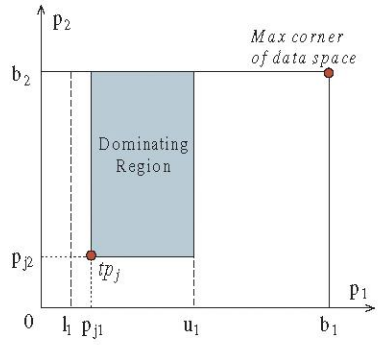


Fig. 6. Dominating Region with Constraints

Suppose the value range on dimension p_k is $[s_k, b_k]$ in terms of all R_is. Then the volume of tuple tp_j's dominating region with respect to constraints is

$$\widetilde{VDR}_j = \prod_{k=1}^{n} (\widetilde{b}_k - p_{jk}) \tag{5}$$

where

$$\widetilde{b}_k = \begin{cases} b_k, & \text{if } C_k = \emptyset \\ u_k, & \text{if } l_k \le p_{jk} \le u_k \\ p_{jk}, & \text{otherwise} \end{cases}$$

Note that it does not make sense to define the dominating region for points out of the region specified by all constraints, because such points does not appear in the skyline result and will not be considered when we select filtering points. In the Formula 5, we ensure the $\widetilde{VDR}$ value is zero for any point uncovered by given constraints, which will prevent it from being chosen as a filtering point.

Given a collection of skyline points $\{s_1, s_2, \ldots, s_k\}$, their dominating regions only differ on their own positions in the data space. Thus for any s_i in that collection, its dominating region is represented by a hyper-rectangle HR_i ($[p_{i1}, \widetilde{b}_1]$, ..., $[p_{in}, \widetilde{b}_n]$), where $\widetilde{b}_i$ is determined as Formula 5 describes.

2) Fused Dominating Region: For multiple filtering points, we need to consider their overall filtering capability which is measured by the volume of their fused dominating region. Given any two distinct skyline points s_i and s_j, the volume of their fused dominating region is

$$\widetilde{VDR}_{i,j} = \widetilde{VDR}_i + \widetilde{VDR}_j - \prod_{k=1}^{n} (\widetilde{b}_k - max(p_{ik}, p_{jk})) \tag{6}$$

Figure 7 shows an example of fused dominating region of two skyline points tp_i and tp_j, where constraint $[l_1, u_1]$ is specified on the dimension p_1.

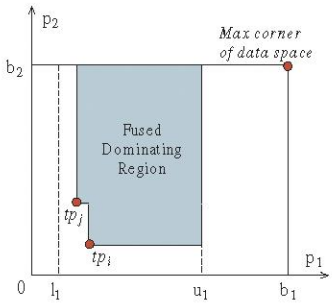


Fig. 7. Fused Dominating Region

By applying the Inclusion-Exclusion principle further, we can compute the volume of fused dominating region of arbitrary number (K) of skyline points.

$$\widetilde{VDR}_{1..K} = \sum_{i=1}^{K} \widetilde{VDR}_i + \sum_{j=2}^{K}((-1)^{j-1} \sum_{1\le i_1<\ldots<i_j\le K} \prod_{k=1}^{n}(\widetilde{b}_k - max(p_{i_1k},\ldots,p_{i_jk}))) \quad (7)$$

We call it *fused $\widetilde{VDR}$ value* for K skyline points. Basically, the computation complexity of the Formula 7 is $\mathcal{O}(2^K)$, which is quite high. What makes it even more complex is that the optimal selection of multiple filtering points is made by choosing points with the maximum volume of fused dominating region. For that purpose, we need to enumerate all $\binom{l}{K}$ K-combinations of SK_{init}, whose computation complexity is $\mathcal{O}((\frac{e\cdot l}{K})^K)$. Hence the total complexity of computing the fused $\widetilde{VDR}$ values for all K-combinations is $\mathcal{O}((\frac{2\cdot e\cdot l}{K})^K)$. This high complexity renders undesirable the optimal selection of K filtering points with the maximum fused $\widetilde{VDR}$ value. Therefore, we turn to alternatives that make more computationally efficient selections at the cost of the quality of the results.

B. Selection of K Filtering Points

We proceed to present two heuristics to guide the selection of multiple filtering points.

1) Heuristic I: Maximal Sum of $\widetilde{VDR}$s: The first heuristic for selecting multiple filtering points is straightforward. It maximizes the sum of the K $\widetilde{VDR}$ values of all possible choices. To accomplish this, we need to sort points in SK_{init} in a non-ascending order and then pick the top-K ones. In an alternative way, the sorting can be integrated into the skyline computation, which produces a sorted SK_{init} for easy picking of top-K points. For both ways, the time complexity depends on the sorting method used, which usually does not go beyond $\mathcal{O}(l^2)$.

This heuristic, MaxSum for short, actually simplifies the computation by ignoring the overlapping between different skyline points' dominating regions. In other words, the overlapping between dominating regions determines the accuracy of this approximation. The smaller the overlapping regions are, the more accurate the method will be.

2) Heuristic II: Maximal Distance Between Points: In the second heuristic we intend to take into account the topology between filtering points, to reduce the overlapping faced by the first heuristic. Distance is a simple metric to help consider this. Intuitively, the farther two skyline points are apart, the less their dominating regions overlap. Hence we propose a greedy heuristic, MaxDist for short, that maximizes the distance between filtering points.

The algorithm of this heuristic is shown in Figure 8. Initially, it picks from SK_{init} two points between which the distance is the largest among all pairs (line 2). Then, it incrementally selects points from SK_{init} and add them to the filtering set, until K filtering points are obtained. In every incremental step, the point with the maximal sum of distances to all current filtering points is selected (line 5). The time complexity of this algorithm is $\mathcal{O}(l^2)$, much less than the exact approach in Section V-A.2.

Algorithm maxDist(SK_{init}, K)
Input: SK_{init} is the initial skyline
K is the number of filtering points needed
Output: a set of multiple filtering points
1. $F_{flt} = \emptyset$;
2. Pick s_i and s_j from SK_{init} satisfying $|s_i, s_j| \ge |s_{i'}, s_{j'}|$, $\forall\ 1 \le s_{i'}, s_{j'} \le l$;
3. $F_{flt} = \{s_i, s_j\}$; $SK' = SK_{init} - \{s_i, s_j\}$;
4. **while** ($|F_{flt}| < K$)
5. Pick s_i from SK' satisfying $\sum_{s_j \in F_{flt}} |s_i, s_j| \ge \sum_{s_j \in F_{flt}} |s_{i'}, s_j|$, $\forall\ s_{i'} \in SK'$;
6. $F_{flt} = F_{flt} \cup \{s_i\}$; $SK' = SK_{init} - \{s_i\}$;
7. **return** F_{flt}

Fig. 8. Algorithm of Heuristic MaxDist

C. Dynamic Update of Multiple Filtering Points

After a site S_i receives a query request with a set of filtering points F_{flt}, it will execute a local query processing. To take advantage of the filtering points, the local processing can be implemented in two ways. In an integrated way, F_{flt} is checked against every candidate point pt met during the skyline computation, any dominated pt is ignored, and any dominated s_i in F_{flt} is removed from the set. In a separate way, a local skyline is computed first, and then it will be compared with F_{flt} to filter out those unqualified candidates and dominated s_is. The integrated way is seamlessly applicable to those centralized skyline algorithms that are not based on data transformation [1], [2], [3], [4], [5], whereas the separate way is applicable to all existing skyline algorithms.

For either way, we get a local skyline result SK_i and the reduced set of filtering points F'_{flt}. Now we need to update the F'_{flt} with points in SK_i, so that the dominating capability of the multiple filtering points is maintained or increased. A simple yet efficient way is to treat all points in these two sets equally, and to pick K ones $SK_i \cup F'_{flt}$ using the heuristics

proposed in Section V-B. To update F'_{flt} based on Heuristic I (MaxSum), SK_i is sorted in non-ascending order of fused $\widetilde{VDR}$ values, and merged with S'_{flt} to get the top-K ones as those new filtering points. To update F'_{flt} based on Heuristic II, the algorithm in Figure 8 is adapted to pick K points from the union of F'_{flt} and SK_i.

VI. Experimental Study

In this section, we evaluate our skyline query mechanism with extensive experiments. We study the performance of our skyline computing approach *PaDSkyline* comparing with **Naive** and **Random** approach. In the Naive approach, a query originator directly sends out its query request to each site, receives results and merge with its local result, till all sites have replied. The naive approach does not execute queries in any specific order of sites, nor does it employ any filtering points. The Random approach is almost the same to our proposal *PaDSkyine* method (with two heuristics **MaxSum** and **MaxDist**), except that it selects filtering points randomly.

We in the experiments use two kinds of synthetic datasets: independent and anti-correlated datasets. The parameters used in the experiments are listed in Table I. Unless stated otherwise, the default parameter values, given in bold, are used. Besides, we use a real-life dataset of NBA players' season statistics from 1949 to 2003[1], which contains 16,644 records of 17 attributes and approximates a correlated data distribution.

Parameter	Setting
Dimensionality	2, 3, 4, **5**
Number of sites	1,000, 2,000, ..., **10,000**
Filter points percentage	10%, 20%, ..., **50%** ..., 90%
Cardinality of each site	**1,000**
Number of queries	**50**

TABLE I
Parameters Used in Experiments

All simulation experiments are conducted on a Linux Server with two Intel(R) Xeon(TM) 2.80GHz processors and 1.0GB of RAM. We consider three performance metrics. First we study **Data Reduction Efficiency**, the efficiency of multiple *filter points* towards the local skyline computing in terms of the data reduction rate DRR, which was proposed in [12]. The DRR is the proportion of data points reduced by the *filtering point* to the number of points in the unreduced skyline. It is defined as

$$DRR = \frac{\sum_{i=1,i\neq org}^{m}(|unredSK_i| - |redSK_i| - K_i)}{\sum_{i=1,i\neq org}^{m}|unredSK_i|}, \quad (8)$$

where K_i is the number of filtering points sent to a processing site, and m is the network size. Second, we consider **Response Time** which records the overall query processing time. Third we investigate **Precision**, which indicates the quality of skyline points returned to the query site for a certain query. Except in the Naive approach, each group head is regarded as a query site for its group members. It is formally defined as ($P = \frac{|A|}{|B|}$), where A is the set of exact skyline points a query site obtains finally, and B is the set of resulting points returned to the query site in the distributed environment. P captures the fraction of relevant points a query processing approach needs to identify and returns to the user.

[1] http://databasebasketball.com

A. Effect of Network Size

In this experiment, we consider the effect of network size on the performance of different methods. Note that, for the Naive approach, it returns all the skyline candidates to the query site, and hence the DRR is zero. Figures 9(a), 10(a) and 11(a) show that MaxSum yields the best DRR performance. Carefully selected filtering points can significantly reduce the data communication when the skyline query is forwarded between sites, e.g. MaxSum approach obtains 80% higher DRR than Random approach. Regarding to MaxSum and MaxDis, there is a tradeoff between sizes of dominating regions and overlap of dominating regions when we select filtering points. From the results, we can see that MaxSum has better filtering capability overall.

Figures 9(b), 10(b) and 11(b) show the results of response time. Clearly, the Naive method is the worst among all four ones, because it requests the local skyline points from all the data sites in the network, which introduces higher communication cost and heavier computational workload in the query originator site to merge all the unfiltered answers. MaxSum and MaxDist perform better than Random because of their stronger filtering capabilities, which result in smaller amount of transferred data and shorter response time. However, the gap is narrowed as random selection of filtering points is computationally faster than two heuristics.

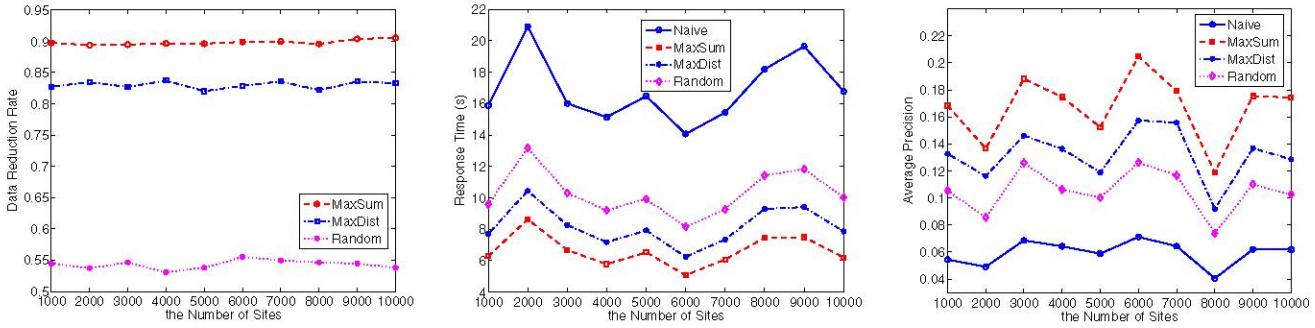
The experimental results of precision, indicator of the quality of skyline points returned to query sites, are reported in Figures 9(c), 10(c) and 11(c). Averagely, MaxSum is about 200% better than Naive and 80% better than Random. This significant performance gain is again attributed to our carefully selected multiple filtering points, which increase the ratio of final skyline points to all points transferred through the network.

As our approaches outperform the competitors in all experimental cases, we in the sequel will show the results on independent dataset only due to the space constraint.

B. Effect of Filtering Points

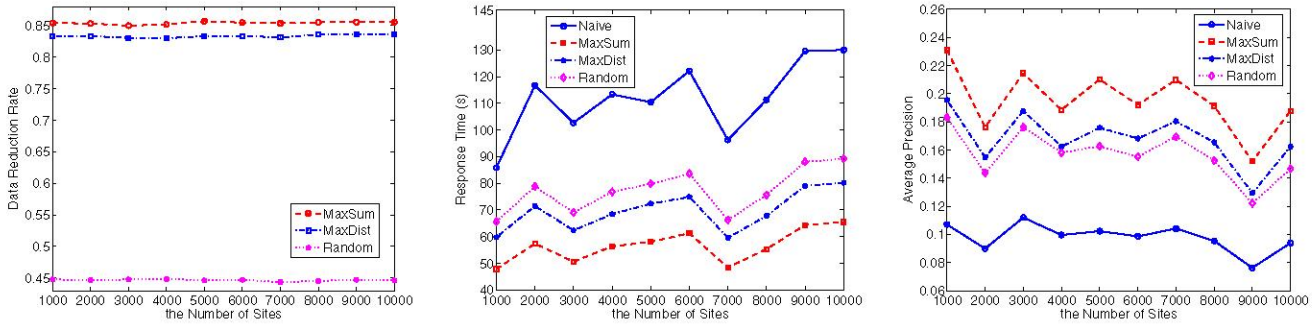
How the performance varies with different number of filtering points is reported in Figure 12. Referring to Figure 12(a), a larger number of filtering points do not necessarily ensure a higher DRR. This is because more filtering points also count in the DRR calculation in Formula 8, and the collective dominating capability of multiple points does not increase enough to offset that effect, especially for the Random approach without any special consideration in points selection.

More filtering points also prolong the response time, as shown in Figure 12(b), because they need more time to



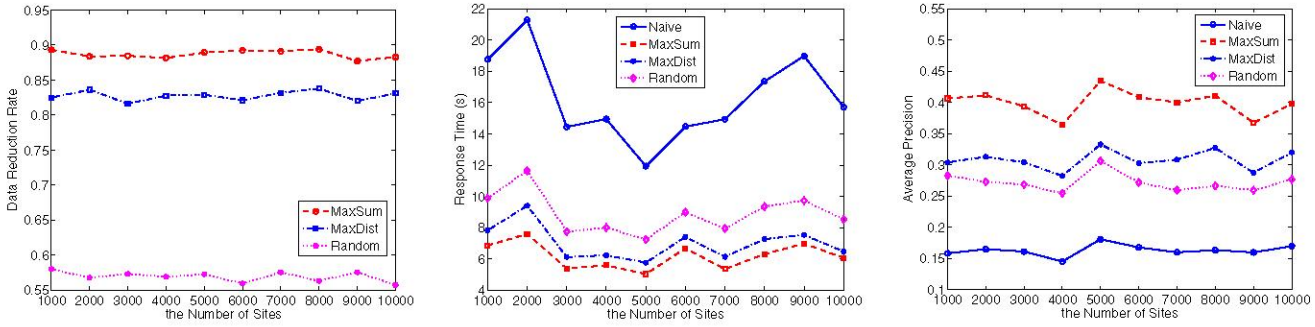
(a) Data Reduction Rate (b) Response time (c) Precision

Fig. 9. Performance on Independent Datasets with Different Network Sizes



(a) Data Reduction Rate (b) Response time (c) Precision

Fig. 10. Performance on AntiCorrelated Datasets with Different Network Sizes



(a) Data Reduction Rate (b) Response time (c) Precision

Fig. 11. Performance on NBA Dataset with Different Network Sizes

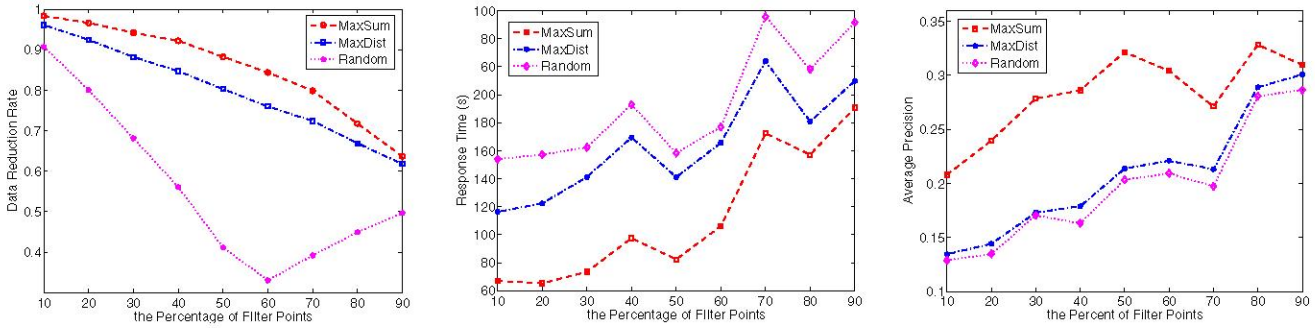
be transferred from site to site through the network. On the other hand, more filtering points increase the average precision, as shown in Figure 12(c), because they do rule out more unqualified intermediate answers (though not enough to maintain and increase DRR) which otherwise are returned to the query sites, i.e. group heads.

C. Effect of Dimensionality

The effect of dataset dimensionality is reported in Figure 13. As the dimensionality increases, DRR decreases for all non-naive approaches as shown in Figure 13(a), and average precision increases for all approaches including Naive as shown in Figure 13(c). The cause for this discrepancy observed is the cardinality of the final skyline, which grows markedly as dimensionality increases [16], [17]. A larger final skyline results in relatively fewer unqualified intermediate answers, which explains the decrease of DRR. A larger final skyline certainly produces more valid points returned to the query site, and thus increasing the average precision. A larger final skyline also causes more data points to be transferred through the network, which leads to the longer response time seen in Figure 13(b).

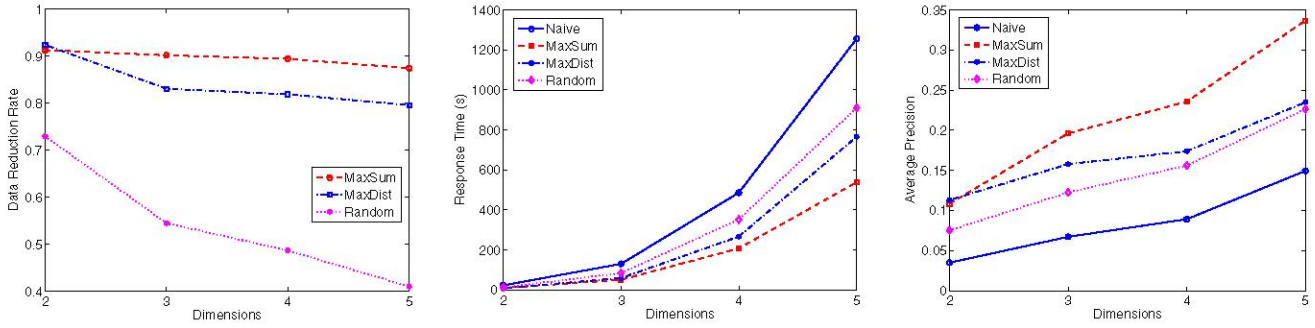
VII. Conclusion

This paper addresses constrained skyline queries in a distributed network environment without overlay structures. First



(a) Data Reduction Rate (b) Response time (c) Average Precision

Fig. 12. Performance with Different Numbers of Filtering Points



(a) Data Reduction Rate (b) Response time (c) Average Precision

Fig. 13. Performance with Varying Dimensionality

a specific partition algorithm is proposed which divides all relevant sites into groups, such that a given query can be executed in parallel among all site groups. For the query execution within a single group of sites, we propose query forwarding plans between sites and designate a group head responsible for query results merger. Based on the inter-group and intra-group query execution mechanism, a parallel distributed skyline algorithm is given, together with a cost model for cost estimation. As a substantial enhancement, it is discussed how to select local skylines as filtering points, which are used in distributed query processing to prevent more data from being transmitted through the network. Extensive experiments are conducted, producing results which demonstrate that our proposals are efficient in distributed query processing and robust to scales of both data and network size.

ACKNOWLEDGMENT

This research was supported by the National Natural Science foundation of China under Grant No.60603045, National Grand Fundamental Research 973 program of China under Grant No.2004CB318204.

REFERENCES

[1] S. Borzonyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proc. ICDE*, 2001, pp. 421–430.

[2] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting." in *Proc. ICDE*, 2003, pp. 717–816.

[3] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets." in *VLDB*, 2005, pp. 229–240.

[4] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," in *Proc. VLDB*, 2002, pp. 275–286.

[5] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proc. ACM SIGMOD*, 2003, pp. 467–478.

[6] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *Proc. VLDB*, 2001, pp. 301–310.

[7] G. Hjaltason and H. Samet, "Distance browsing in spatial database," *ACM TODS*, vol. 24, no. 2, pp. 265–318, 1999.

[8] W.-T. Balke, U. Güntzer, and J. X. Zheng, "Efficient distributed skylining for web information systems," in *Proc. EDBT*, 2004, pp. 256–273.

[9] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi, "Parallelizing skyline queries for scalable distribution." in *Proc. EDBT*, 2006, pp. 112–130.

[10] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM*, 2001, pp. 161–172.

[11] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu, "Efficient skyline query processing on peer-to-peer networks," 2007, pp. 1126–1135.

[12] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi, "Skyline queries against mobile lightweight devices in MANETs." in *Proc. ICDE*, 2006, p. 66.

[13] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, Eds., *Mobile Ad Hoc Networking*. New Jersey: Wiley-IEEE Press, 2004.

[14] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. James B. Rothnie, "Query processing in a system for distributed databases (sdd-1)," *ACM TODS*, vol. 6, no. 4, pp. 602–625, 1981.

[15] C. T. Yu and C. C. Chang, "Distributed query processing," *ACM Computing Surveys*, vol. 16, no. 4, pp. 399–433, 1984.

[16] S. Chaudhuri, N. N. Dalvi, and R. Kaushik, "Robust cardinality and cost estimation for skyline operator." in *Proc. ICDE*, 2006, p. 64.

[17] P. Godfrey, "Skyline cardinality for relational processing." in *Proc. FoIKS*, 2004, pp. 78–97.