Efficient subspace skyline query based on user preference using MapReduce

著者	LI Yuanyuan, LI Zhiyang, DONG Mianxiong, QU Wenyu, JI Changqing, WU Junfeng
journal or publication title	Ad Hoc Networks
volume	35
page range	105-115
year	2015-12
URL	http://hdl.handle.net/10258/00008596

doi: info:doi/10.1016/j.adhoc.2015.07.006

Efficient Subspace Skyline Query based on User Preference using MapReduce

Yuanyuan Li[a,b], Zhiyang Li[*,a], Mianxiong Dong[c], Wenyu Qu[a], Changqing Ji[d], Junfeng Wu[a,e]

[a] *School of Information Science and Technology, Dalian Maritime University, Dalian, China*
[b] *School of Software, Dalian Jiaotong University, Dalian, China*
[c] *Department of Information and Electronic Engineering, Muroran Institute of Technology, Hokkaido, Japan*
[d] *College of Physical Science and Technology, Dalian University, Dalian, China*
[e] *School of Information Engineering, Dalian Ocean University, Dalian, China*

Abstract

Subspace skyline, as an important variant of skyline, has been widely applied for multiple-criteria decisions, business planning. With the development of mobile internet, subspace skyline query in mobile distributed environments has recently attracted considerable attention. However, efficiently obtaining the meaningful subset of skyline points in any subspace remains a challenging task in the current mobile internet. For more and more mobile applications, subspace skyline query on mobile units is usually limited by big data and wireless bandwidth. To address this issue, in this paper, we propose a system model that can support subspace skyline query in mobile distributed environment. An efficient algorithm for processing the Subspace Skyline Query using MapReduce (SSQ) is also presented which can obtain the meaningful subset of points from the full set of skyline points in any subspace. The SSQ algorithm divides a subspace skyline query into two processing phases: the preprocess phase and the query phase. The preprocess phase includes the pruning process and constructing index process which is designed to reduce network delay and response time. Additionally, the query phase provides two filtering methods, SQM-filtering and ε-filtering, to filter the skyline points according to user preference and reduce network cost.

*Corresponding author
Email address: lizy0205@dlmu.edu.cn (Zhiyang Li)

Extensive experiments on real and synthetic data are conducted and the experimental results indicate that our algorithm is much efficient, meanwhile, the pruning strategy can further improve the efficiency of the algorithm.
Key words: `subspace skyline query`, `MapReduce`, `pruning strategy`, `grid`, `user preference`

1. Introduction

Skyline query has attracted an increasing amount of attention over the past few years. Skyline query has been widely applied in highly mobile distributed environments for multiple-objective decisions. For example, a tourist may search for a suitable hotel using a mobile phone network when he/she arrives at the airport of a city.

Due to large-scale data and high processing costs, it is not possible to compute the skyline points using terminals such as mobile phones. The number of skyline points quickly increases when the amount of data increases, particularly for increasing trend of applications to deal with big data, for example, big data on the Social network. To address this problem, skyline computation in a distributed environment is the best option. [1, 2, 3, 4] proposed some approaches for skyline processing in P2P network. These methods are restricted to their own specific scenarios, such as overlay network, and cannot be adapted for the aforementioned scenario.

Furthermore, skyline query can be computationally expensive when it provides numerous candidate attributes. In many applications, various users may focus their attention on different subsets of the attributes according to their own interests. [5, 6] proposed efficient skyline algorithms in subspaces that can retrieve the skylines in a user-defined subset of attributes. MapReduce framework [7] is a programming model for processing large datasets using a distributed, parallel algorithm on a cluster. A considerable amount of work has been conducted to migrate traditional skyline algorithms to the MapReduce framework, such as [8, 9, 10, 11], but they cannot support subspace skyline query.

The subspace skyline query assumes every attribute to be of equal importance, and the difference in the importance of attributes is not considered. Bias to some attributes of greater interest was considered in [12], in which the authors proposed the Weighted Dominant Skyline. However, it is not possible for users to provide the weight assignment without any initial knowledge of the dataset. In this paper, we only require users to provide the ranks of their attributes of interest.

Table 1: Dataset of Hotels

ID	Price (RMB)	Mileage (KM)	Occupancy Rate
p_1	200	7	0.5
p_2	200	3	0.6
p_3	250	5	0.8
p_4	500	3	0.7
p_5	150	9	0.6
p_6	200	9	0.5
p_7	300	6	0.7
p_8	500	5	0.8
p_9	300	7	0.9

Taking Table 1 as an example, a dataset P=$\{p_1, p_2, ..., p_9\}$ about hotels contains 3 attributes: the price, the distance to the airport (Mileage) and the occupancy rate. Assuming that the attributes are of equal importance, there are 4 skyline points in P: p_1, p_2, p_5 and p_6. However, in many cases, the relative importance of attributes are often different. For example, when a tourist is very tired, the distance to the airport is crucial for him/her, and the other attributes are secondary. In this case, although p_5 and p_6 are skyline points, the tourist does not consider these points because the distance to the airport is too far. In the above example, the interesting subset of skyline points is $\{p_1, p_2\}$.

The aim of a skyline query is to help users manually make a decision. As the size of the dataset increases, the size of skyline points becomes too large.

There are many meaningless skyline points to report, such as p_5 and p_6 in the above example. In this paper, we present a new algorithm which we call Subset Skyline Query or SSQ for short to return the meaningful subset of subspace skyline points.

The major contributions of this work are summarized as follows: (1) We present a system model for implementing the skyline operator using MapReduce in any subspace, in which the requirements of mobile internet and big data are carefully considered. (2) We propose the algorithm of the Subspace Skyline Query (SSQ) which can obtain the meaningful subset skyline of skyline points. In the first phase, we design a pruning strategy based on grid to reduce the network delay and response time. It can effectively prune out non-skyline points in advance. The constructing index process is also used to support subspace skyline query. In the second phase, we provide two filtering methods, called SQM-filtering and ε-filtering, to filter the skyline points according to user preference and reduce network communication. (3) We conduct experiments on real and synthetic data. Experimental evaluations show that SSQ can significantly improve the efficiency of the subspace skyline query over big data.

The remainder of this paper is organized as follows. In Section 2, we review the previous work related to skyline query processing. Section 3 provides the useful preliminaries. In Section 4, a system model is presented. Section 5 describes the implementation of the SSQ algorithm in the parallel programming framework of MapReduce. Section 6 presents experimental evaluations that demonstrate the efficiency of the proposed algorithm. Finally, we conclude the paper with a summary of our results in Section 7.

2. Related work

The skyline operator is useful for extracting interesting data points from a dataset. Over the past decades, a considerable number of research works have reported on the skyline operator and its variants. Börzsönyi et al. [13] first introduced the skyline operator into the relational database and proposed

two algorithms: Block Nested Loop (BNL) and Divide-and-Conquer (D&C). Chomicki et al. [14] presented Sort-Filter-Skyline(SFS) as a variant of BNL, which can immediately eliminate objects dominated by others in the presorted dataset. The primary shortcoming of the above algorithms is their dependence on memory capacity. Many skyline query algorithms based on indexing have been proposed, such as the nearest neighbor [15] and branch-and-bound skyline [16] algorithms. Due to the dimension curse, the size of the skyline is typically large. Consequently, many variants of skylines have been proposed. Subspace skyline query were first discussed by Pei, J et al. [5] and subsequently elaborated in later works [6]. The key concept of a subspace skyline query is to implement the skyline operator in the most interesting subset of all dimensions by the user. Clearly, skyline query processing using the index structure mentioned in [15, 16] over all dimensions, such as R-tree, cannot support subspace skyline query. All of these works assumed that skyline query is performed on centralized systems. Unfortunately, a large amount of data can result in low efficiency for skyline query performed in a centralized setting.

Deviating from skyline query on centralized systems, significant research effort has been devoted to implementing the skyline operator in distributed environments. Skyline queries in P2P networks were discussed in [2, 3, 4], in which unstructured peers or routing indexes were used to identify relevant peers. Skyline processing has also been studied in other distributed environments, such as web information systems [17, 18, 19]. They are not adapted for our scenario because skyline processing over big data cannot be performed in lightweight terminals. Motivated by the fact that many points which belong to the local skyline sets do not belong in the final skyline result set, to reduce the amount of local skyline communication, in [20, 21], the authors presented space partitioning schemes such as the angle-based partitioning approach. In contrast to the parallel skyline algorithms designed for share-nothing distributed environments by only exchanging messages, [9, 10, 22, 11] focused on exploiting database programming models, such as MapReduce, but these models cannot support subspace skyline query. Our method aims to effectively support subspace sky-

line query based on user preferences for large-scale data.

In the above studies, the conventional skyline algorithms were either not suitable for large-scale data or did not support distributed subspace skyline query. Furthermore, with the development of mobile internet devices, skyline query in both mobile and distributed environments has become an important problem. The distributed nature of the environment makes the task of discovering skylines even more challenging, particularly in any subspace. Conventional skyline algorithms do not support subspace skyline query in distributed environments. As the amount of data greatly increases, the number of skyline points also increases; unfortunately, many of the returned skyline points do not meet the user preferences. In addition, because some useless skyline points are reported, it is difficult to for the user to perform a manual evaluation. Because the wireless bandwidth is scarce, it is not necessary to report all of the skyline points to the mobile terminal.

In this paper, we focus on how to perform distributed skyline query to retrieve the meaningful subset of skyline points from the full set of skyline points according to user preference. For this purpose, we present a system model of Subspace Skyline Query in mobile and distributed environments. In SSQ, a pruning strategy is proposed to reduce network communication and to minimize the response time. To support subspace skyline query, an index over the distributed data was constructed in SSQ. Additionally, SSQ incorporates relative attribute importance into skyline query and provides two filtering methods to remove the data points whose values are worse in the important attributes. The new SSQ algorithm can report the meaningful subset of the full skylines in any subspace.

3. Preliminaries

In this section, we provide some notations that will be used throughout the paper and the rationale of both the SQM filtering method (SQM-filtering) and the ε filtering method (ε-filtering). These filtering methods consider differences

in the importance of the attributes.

Given a d-dimensional space S=$\{s_1, s_2, ..., s_d\}$, a set of points P=$\{p_1, p_2, ..., p_n\}$ is said to be a dataset on S if every $p_i \in P$ is a d-dimensional data point on S.

Definition 1. *(Dominate) Given a dataset P on S, a point $p_i \in P$ dominates another point $p_j \in P$ if it is better than or equal to p_j in all dimensions and better than p_j in at least one dimension.*

Definition 2. *(Skyline) A point p_i is a skyline point on S if and only if there does not exist a point p_j dominating p_i in P.*

Definition 3. *(Subspace Skyline) A subset of S, $F \subseteq S$ forms a k-dimensional subspace where $k = |F|$ and $k \leq d$. For a point p_i in space S, the projection of p_i in subspace F is denoted by p'_i, which is a k-tuple. The p'_i is a subspace skyline on subspace F if and only if there does not exist a point p'_j dominating p'_i on the subspace F.*

Definition 4. *(Partial Order Relationship of Attribute Importance) We assume that there exists a binary relation denoted by $\succ$ on the subspace F of S. Here, $\succ$ can be a '$>$' relationship according to the importance of different attributes. If f_1 is more important than f_2 according to user preference, where $f_1 \in F$ and $f_2 \in F$, we use $f_1 \succ f_2$ to denote the partial order relationship. We obtain the order of k-dimensional subspace F, which is $\{f_1, f_2, ..., f_k\}$.*

Example 1. *Considering the 3-dimensional subspace $F = \{Price, Occupancy\ rate, Mileage\}$ in Table 1. The order relationship of the attributes is $Mileage \succ Price \succ Occupancy\ rate$ according to the tourist's preference.*

For simplicity and without loss of generality, we assume that the smaller is better. The objective functions of the multiple objective optimization problem can be formulated as $\min(f_1(x), f_2(x), ..., f_k(x))$, $x \in P$, where $f_i(x)$ is the value of the i^{th} dimension in the tuple x.

$$R_1 = arg \max_{x \in R_0} f_1(x) \tag{1}$$

$$R_2 = arg \max_{x \in \tilde{R}_1} f_2(x) \tag{2}$$

... ... (3)

$$R_k = arg \max_{x \in \tilde{R}_{k-1}} f_k(x) \tag{4}$$

We use Eq. (1) to compute the worst tuples for the first dimension as a set R_1. In Eq. (1), R_0 is the initial dataset (i.e., P). Subsequently, we obtain the relatively optimal tuples in the first dimension as $\tilde{R}_1$, $\tilde{R}_1 = R_0 - R_1$. We use Eq. (2) to retrieve the worst tuples for the second dimension under the constraint of $\tilde{R}_1$ as R_2, and so on, until we retrieve $\tilde{R}_k$, where $\tilde{R}_k = R_{k-1} - R_k$.

To avoid the case that the size of the result set becomes too large, a tolerant filtering method(ε-filtering) is proposed. It is not necessary to retrieve the worst tuples for the previous attribute because it retrieves the worse tuples as R_i for the current dimension under a certain tolerant constraint ε_i. In Eq. (5), ε_i is tolerance limit of i-th attribute, and it may be given based on knowing the user preference in advance. Consequently, we can obtain more elements in R_i to satisfy the condition in Eq. (5) when $0 \leq \varepsilon_i \leq 1$. Meanwhile, the relatively optimal answer set becomes small.

$$R_i = \{x | f_i(x) \geq \varepsilon_i max(f_i(x)), x \in \tilde{R}_{i-1}\} \tag{5}$$

Example 2. *By applying the above equations to the dataset of Table 1,* $R_1 = \{p_5, p_6\}$, $\tilde{R}_1 = R_0 - R_1 = \{p_1, p_2, p_3, p_4, p_7, p_8, p_9\}$, *where* $\varepsilon_1 = 1$. *Assuming that the distance limit (and price or occupancy rate limits) depends on the value of* ε_1 (ε_2, ε_3), *then* $R_1 = \{p_5, p_6, p_1, p_9\}$, $\tilde{R}_1 = \{ p_2, p_3, p_4, p_7, p_8\}$, *where* $\varepsilon_1 = \frac{7}{9}$.

4. SYSTEM MODEL

As shown in Figure 1, this section presents a framework that is applied to subspace skyline query under a mobile distributed environment. It is a

client/server architecture.

The client is an application that runs on terminals such as personal digital assistants and cell phones. These terminals can communicate with servers using networks or wireless channels and receive the query results. Users can use these terminals to select the preference of attributes and rank them with different importance. Then, a client submits a skyline query to a server. The sever may consist of many nodes that are organized by a cloud computing platform [23] such as Hadoop. Finally, the skyline algorithm is executed in this environment.

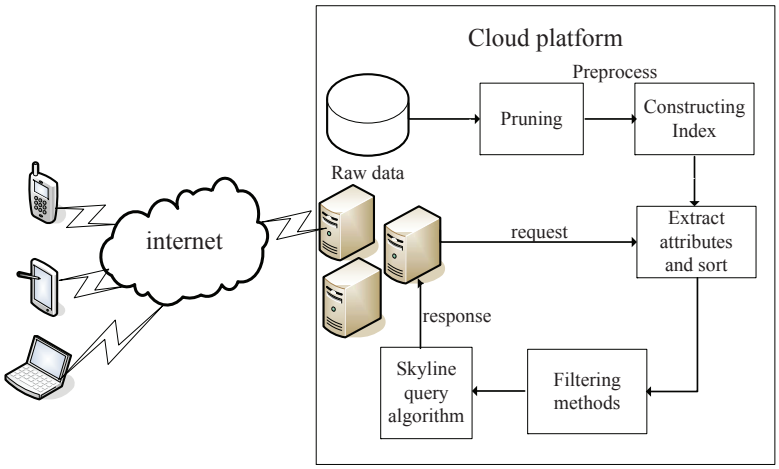


Figure 1: System model of Subspace Skyline Query

The system model consists of two modules to compute skylines: the preprocess module and the query module. In the former module, the system scans the distributed data into a regular grid and prunes the data points in those cells dominated by others. This pruning strategy aims to reduce the costs of constructing an index over big data and skyline query processing. Subsequently, the data points are sorted for each dimension according to user preference. k sorted sets (k is the number of dimensions) are combined into an index file. The aim of constructing the index is to support skyline query in arbitrary subspace. After receiving a query request, the query module can extract the subset of attributes

from the index of the above module according to the user's interest. Next, the filtering method is executed in the subspace of interest to the user. The query results are returned to the terminals by mobile internet. Moreover, these results must be manageable because they need to be displayed on terminals for manual evaluation. The filtering methods can filter out the skyline points whose values in the important attributes are worse.

The novelty of our system model is to propose a Lightweight Client Framework in a distributed environment. Skyline computing can be performed in back-end servers. First, it can be applicable to skyline query in mobile internet environments. Second, it improves the efficiency of constructing the index file because the pruning strategy can reduce the amount of transferred data. Furthermore, as a preprocessing step, pruning cannot occupy the running time of the skyline query. Finally, it incorporates multiple-objective decision into the query process and provides the SQM-filtering and the ε-filtering methods. We will describe the specific approaches for how to compute the skylines using the above two filtering methods and return the meaningful subset of skylines in detail in Section 5.

5. Subspace Skyline Query

In this section, we first present the grid-based pruning strategy, then we introduce the process of constructing the index file using MapReduce. Finally, we provide a more detailed discussion on the subspace skyline query.

5.1. Grid-based Pruning Strategy

Skyline operations are expensive when large-scale datasets are encountered. To reduce the amount of transferred data for distributed skyline query processing, a grid-based data summary was proposed in [24, 25], that captures the data distribution on each server. What is different from the above work is that we proposed a pruning strategy based on grid. Because the skyline result set is considerably smaller than the original dataset, the pruning strategy can eliminate non-skyline points in advance.

We partition the dataset using a spatial structure named grid, which divides the space or data into a series of contiguous cells and can be assigned unique identifiers [26, 27]. During pruning, the data space is divided into many regular cells of a grid. The data points in each cell are stored as ⟨ key, value ⟩ pairs, such as $\langle c_2, \{p_3, p_4\}\rangle$. We can remove the data points in those cells that are dominated by others according to the positional relationship of the cells. The data points after pruning are used as the input for the skyline query.

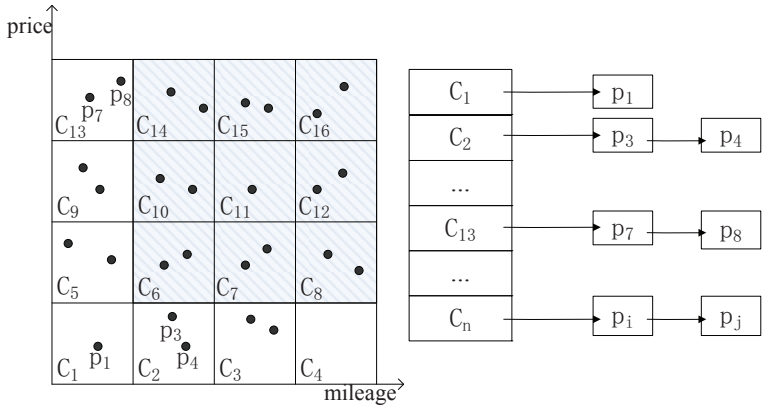


Figure 2: Grid-based pruning strategy

The rationale behind our pruning strategy is that each cell dominates the regions located at the right upper corner. In other words, they have larger values in all dimensions, but for simplicity, the smaller is better. Taking Figure 2 as an example: cell c_1 dominates cells c_6, c_7, c_8, etc. The cells dominated by others can be eliminated along with the points in those cells. There are cells that contain no points, such as c_4, and thus can be eliminated. The points that remain after pruning are organized as a type of ⟨key, value⟩ pair, where key is the id of the cell and value is a list of tuple ids. Although Figure 2 only shows a 2D space, the proposed technique can be applied to higher dimensional spaces.

Algorithm 1: Grid-based Pruning Algorithm (P, λ_i)

initialize set R_1 =P, T=$\emptyset$
initialize set after pruning R_2=$\emptyset$
for (each point $p \in R_1$) **do**
    point p mapped into a coordinate of cell:
    $Intkey.x_i = \left\lfloor \frac{p.x_i}{\lambda_i} \right\rfloor$
insert p into T
GridHash.put ($Intkey$, T)
for (GridHash.Iterator.hasNext()) **do**
    if($Intkey_1$ is dominated by $Intkey_2$)
    GridHash.remove($Intkey_1$)
for (each Point in GridHash) **do**
    OutputCollector.collect($Intkey$, T)
    GridHash.remove($Intkey_1$)
for (each $p \in T$) **do**
    insert p into R_2
emit R_2

Definition 5. *(Grid) Given a d-dimensional dataset P that contains a set of data points. Each point $p \in P$ is represented by $\{x_1, x_2, ..., x_d\}$. The extent of each cell on each dimension is λ_i. In this paper, we assume that the points of data set is evenly partitioned into cells. Cell $Intkey_j$ indicates the cell at space coordinate ($Intkey_j.x_1, Intkey_j.x_2, ..., Intkey_j.x_d$), which can be calculated by $Intkey_j.x_1$=$\left\lfloor \frac{p.x_1}{\lambda_1} \right\rfloor$, $Intkey_j.x_2$= $\left\lfloor \frac{p.x_2}{\lambda_2} \right\rfloor$,..., $Intkey_j.x_d$= $\left\lfloor \frac{p.x_d}{\lambda_d} \right\rfloor$. Clearly, any point p can fall into a cell.*

The selection of λ_i depends on the value domains of the different dimensions. For example, assuming that the distance to the hotel in the city ranges from 1km to 50 km; in this case, threshold values between 2 and 10 should be selected such that the data points are distributed as evenly as possible in each interval.

Depending on the sizes of different datasets, we can adjust the values of λ_i. If the data points in the dataset are scarce, the values of λ_i should be slightly larger. If the data points are numerous, the values of λ_i should be slightly smaller. The values of λ_i are given as the input parameters of Algorithm 1. The coordinate of each grid ($Intkey_i$) is represented by the point located at the lower-left corn in the grid. Taking 2-dimensional space as example, $Intkey_1$ dominates $Intkey_2$ when $Intkey_1.x < Intkey_2.x$ and $Intkey_1.y < Intkey_2.y$. The effect of the cell size on the pruning power will be described in detail in Section 6. The comparisons of grid dominance relationships during pruning are parallelized in distributed systems. The pseudo-code of the pruning algorithm using MapReduce is shown in Algorithm 1.

We analyze the theoretical computation of algorithmically complexity about the pruning strategy. Give a data set D=$\{d_1, d_2, ..., d_\alpha\}$ with n-dimensional attributes. Most of the traditional skyline algorithms have a worst-case complexity of $O(n\alpha^2)$. We partition D into the n-dimensional grid space and the grid has $e_1 \times e_2 \times ... \times e_n$ cells, where $e_1 = \frac{\max\{d_x^1\}}{\lambda_1}$, $e_2 = \frac{\max\{d_x^2\}}{\lambda_2}$, ..., $e_n = \frac{\max\{d_x^n\}}{\lambda_n}$. The cost of grid is negligible compared to computing with all the points and the computing of grid is in the preprocess phase. If the points are assumed to be evenly mapped into all cells, and the pruning strategy returns γ ($1 \leq \gamma \ll e_1 \times e_2 \times ... \times e_n$) cells, the complexity of algorithm is computed as $O(n \times (\frac{\alpha}{e_1 \times e_2 \times ... \times e_n} \times \gamma)^2)$=$O((\frac{\gamma}{e_1 \times e_2 \times ... \times e_n})^2 \times n\alpha^2)$. Obviously, the algorithm after pruning is more efficient.

5.2. Construction of the Index File using MapReduce

Figure 3 shows an example of the input/output with the map and reduce functions in the two MapReduce jobs.

In the first MapReduce job, each map task reads a file split that is a subset of P. The map functions read the input records and separate each dimension from the records to output $\langle$ number of dimensions,(value of the dimension, id of record)$\rangle$ as key/value pairs. Then, data points that belong to the same key will be shuffled to one reduce task. Finally, the values of the same dimension

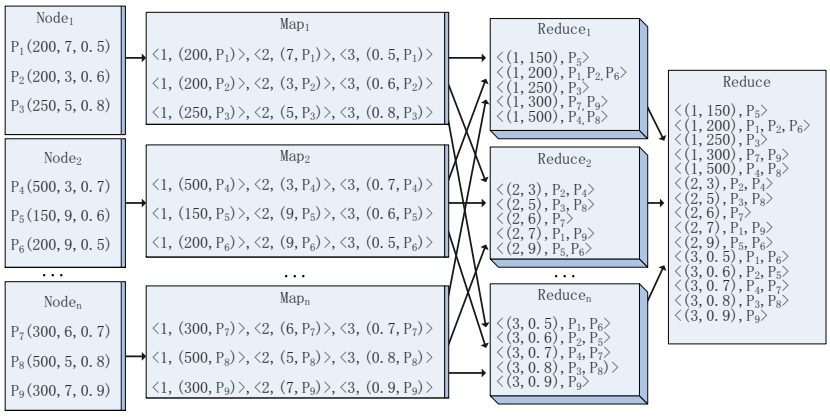


Figure 3: Process of constructing the index file using MapReduce

are transferred to the same reducer, which combines to output ⟨ (the number of dimensions, value of the dimension), list of record ids ⟩. In the second MapReduce job, we launch map functions not to do anything and launch one reduce function to combine the results of the first MapReduce job into an index file. This procedure of preprocessing requires a considerable amount of extra time, but it can be favorable for skyline computing in subspace by extracting the subset of attributes of interest from all of the dimensions.

5.3. Subspace Skyline Query

Figure 4 shows the processing of a skyline query in any subspace. The MapReduce jobs mentioned in Figure 3 transform the input key/value pairs into an index file. Subsequently, a worker scans the entire index into a type of data structure called a tree. The tree consists of two parts: the key and the value. The key is the value of each dimension. The value is an id list of records with the same value in each dimension, which are enclosed with braces as shown in Figure 4 (a).

We extract the subset of all of the attributes of interest to users from an index file. The SSQ algorithm incorporates relative attribute importance into skyline query processing. We need to adjust the order of dimensions according

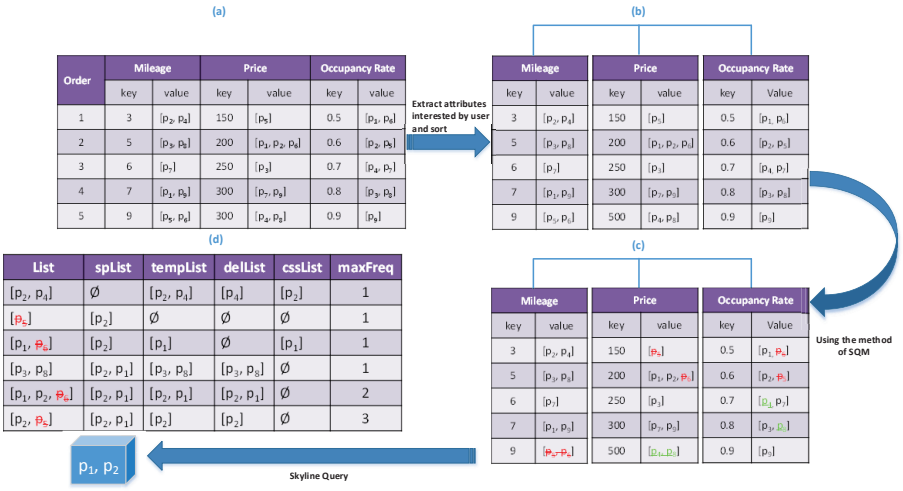


Figure 4: The Subspace Skyline Query

to the rank of each dimension by importance. The importance rank of each dimension stems from the user's request. In other words, the more important a dimension is, the higher priority it should be given. Taking Table.1 as an example, according to the tourist's preference, three dimensions of hotels are selected, and the order of these dimensions is mileage, price, and occupancy rate, as shown in Figure 4 (b).

In general, data points contained in the skyline results will be not a good choice for users. Taking Table 1 as an example, p_5 is a skyline point, but the distance to the airport is too far. Therefore, this point should not be returned to users. Inspired by the concept of multiple-objective decisions, we propose using the SQM filtering method to solve the problem. Based on Eq. (1) to Eq. (4) mentioned in Section 3, the process of filtering is illustrated in Figure 4 (c).

In the above example, the multiple-objective optimization is formulated as min(mileage, price, occupancy rate); assuming that the smaller is better, then $R_1 = \{p_5, p_6\}$. We can prune p_5 and p_6 from R_0 because their values are worse in the mileage attribute. In Figure 4 (c), p_5 and p_6 are marked in red strikeout. When the price is more important than the occupancy rate for the tourist, p_4

and p_8 can be pruned away, and so on. In Figure 4 (c), p_4 and p_8 are marked in green underline. In the skyline algorithm, the data points are not scanned and compared, as shown in Figure 4 (d).

Table 2: Symbols and Definitions

Symbols	Definitions
$tree_i$	the i-th data structure:key and value
key	the value of the data point in each dimension
$value$	id list with the same value in each dimension
$tempList$	the temporary set of data points in the same list
$delList$	the set of data points to be deleted
$cssList$	the difference set of $tempList$ and $delList$
$splList$	the set of skyline points
$maxfreq$	the maximum number of points to be scanned in a list

In the SQM-filtering method, the size of filtered points is limited. In the ε-filtering method, the points that should be removed and the number of these points is dependent on ε_i. ε_i, which was mentioned in Eq. (5), is the tolerance limit of each attribute, and its value can be set according to user preference. Any user has maximum thresholds, and if these maximum thresholds are not met, then we need to filter out these tuples. A user preference is modeled as a set of thresholds along each attribute$\{UP_j^1, UP_j^2, ..., UP_j^k\}$. Therefore, ε_i can be calculated as follows:

$$\varepsilon_i = \frac{UP_j^i}{\max\{f_i\}} \tag{6}$$

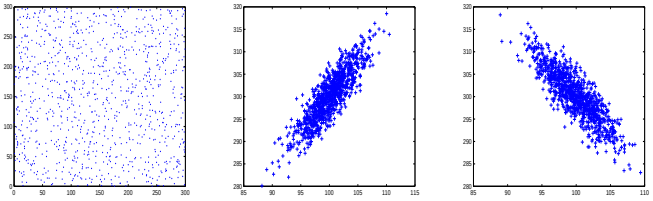
Here, UP_j^i represents the i-th attribute threshold of the j-th user and $\max\{f_i\}$ denotes the maximum of the i-th attribute. They can be given by the users. We assume that the user is willing to pay up to RMB 200 a night and wants a hotel that is located at most 5 km from the airport. Thus, $\varepsilon_1 = \frac{2}{3}$ and $\varepsilon_2 = \frac{5}{9}$. As indicated by Eq. (5), we can eliminate those points that are not of interest to users because their values in the important attributes can not meet the

user's hard constraints. We will argue that the SQM-filtering and ε-filtering can improve the performance of the skyline algorithm in the subsequent sections, particularly when the distribution of data is anti-correlated because in large-scale datasets, there are many of those points in anti-correlated datasets.

We introduce some symbols and definitions in Table 2 to prepare for Algorithm 2. The pseudo-code of the subspace skyline query algorithm using MapReduce is shown in Algorithm 2.

Algorithm 2: Subspace Skyline Query(R_2,$thresholds$)

```
1  initialize set R=R_2 // output of algorithm1
2  initialize set sklList=∅
3  for (each point p ∈ R) do
4      insert p.d_i to tree_i
5  initialize any p.freq=0, maxfreq=0
6  eliminate the points from each tree_i by SQM-filtering or ε-filtering
7  for (any list x_i in tree_1, tree_2, ···, tree_k) do
8      for (j=1; j ≤ k; j++) do
9          tempList =tree_j.x_i
10         for (each point p, q ∈ tempList ∪ sklList) do
11             if ((p.freq>0)&&(p is dominated by q)) then
12             insert p to delList
13         for (each point p ∈ tempList ) do
14             p.freq++
15             if(p.freq > maxfreq) then
16             maxfreq=p.freq
17         cssList = tempList − delList
18         sklList = sklList ∪ cssList
19         if (maxfreq==k) then
20         Break
21     emit sklList
```



(a) independent (b) correlated (c) anti-correlated

Figure 5: Data Distributions

6. EXPERIMENTAL

In this section, we experimentally evaluate the efficiency of the proposed subspace skyline algorithm, and we also discuss its pruning capabilities. All algorithms mentioned in Section 5 are implemented with MapReduce in Hadoop. In addition, the existing parallel algorithm MR-BNL [9] is used as a reference to evaluate the efficiency of our algorithm. In the SSQ algorithm, we implement the two filter methods mentioned above and compare their experimental results.

All experiments were performed on a homogeneous cluster consisting of 12 nodes. Each node had one AMD Opteron 2212 2.00 GHz dual-core processor, a 80 GB SCSI hard drive and an Intel 82551 10/100 Mbps Ethernet Controller. Each machine is connected to a 100 Mbps Ethernet switch and runs the version of Hadoop 0.20.2. On each node, we installed the Ubuntu 10.10, 64-bit, server-edition operating system. One TaskTracker and DataNode daemon ran on each slave node. A single NameNode and JobTracker ran on the master node. The DFS chunk size was set to 64 MB, and 2 GB of memory was allocated for each Hadoop daemon.

We used a real dataset in our experiments. This dataset contains 4887 hotels from a city. We also used synthetic datasets with three different distributions, which are correlated, independent and anti-correlated, as shown in Figure 5.

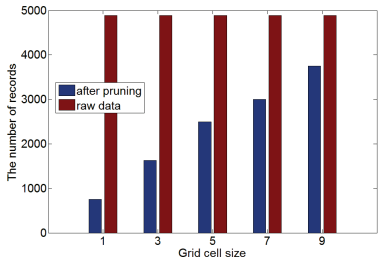


Figure 6: Vary the cell size for pruning

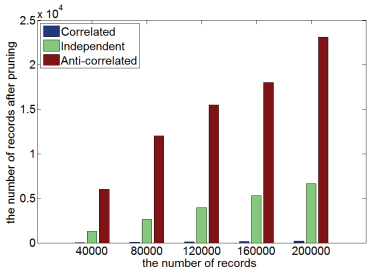


Figure 7: Pruning on different datasets

These three types of datasets are used to evaluate the performance of the algorithms. The number of synthetic datasets varied from $20,000$ to $500,000,000$, and the size of the datasets ranged from 269 KB to 6.42 GB.

The first set of experiments evaluate the pruning capabilities on the real dataset. We examine the impact of the size of the grid cell on the pruning capabilities. We use the real dataset with 4887 records when the size of the grid cell varies from 1 to 9. The results are shown in Figure 6. As shown in this figure, the smaller the grid cell size is, the greater is the pruning efficiency. When the size of the grid cell is set to 1, the number of records after pruning is only 752, which is approximately 1/7 of the raw data.

To evaluate the effect of the data distribution on the pruning capabilities, we use the synthetic datasets. The number of points ranges from $40,000$ to $200,000$. Figure 7 presents the effects of different distributions on the pruning capabilities when the grid cell size is set to the same value. The results indicate that the pruning capability is sensitive to the data distribution. We observe that the pruning efficiency for correlated datasets scales better than the others. This result is because in correlated datasets, the points are densely distributed in the grids, which are dominated by each other. Therefore, we can obtain only 9 records after pruning when the dataset contains $40,000$ records. Similar results were observed for uniformly distributed datasets. Evidently, for the anti-correlated datasets, the performance of pruning is worse than for the above two data distributions. Because in anti-correlated datasets, the points

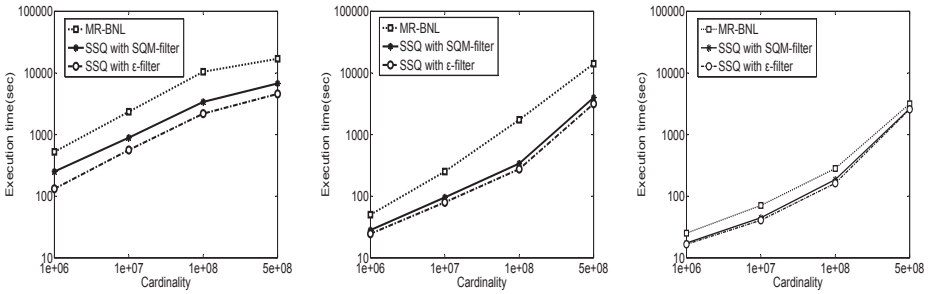
are distributed in the cells that are rarely dominated by each other. The experiment results show that the pruning algorithm can prune the redundant points effectively and improve the query efficiency.

Table 3: Running time on synthetic dataset ($4e8$ records)

Data distribution	TPG(sec)	TCI(sec)	TSSQ(sec)	Total Time(sec)
Correlated	3715.4	523.7	4.4	4243.5
Independent	3266.5	2156.4	120.2	5543.1
Anti-Correlated	3629.5	2814.6	299.1	6743.2

In the following experiments, we use the synthetic datasets with $400,000,000$ points to further analyze the execution time of SSQ on datasets with different distributions. The cost of the proposed algorithm consisted of three parts, which are shown in Table 3. TPG is the time for constructing the inverted grid index. TCI is the time for pruning by comparing the dominance relationship of cells. TSSQ is the response time of the subspace skyline query. The TSSQ is clearly very short compared to TPG and TCI. This result occurs because it must scan the entire dataset to cells. When the scale of the dataset is very large, the TPG cost is high. TCI is the running time for pruning and constructing the index, which is shorter than the first process. Because the subspace skyline query process only scans a part of the index file to acquire the results, TSSQ has a lower cost. The results show that the correlated dataset has the fastest response time compared with the other two data distributions, which occurs because the correlated dataset has the most powerful pruning capabilities and removes more non-qualified records.

Next, we compare the overall execution time of SSQ on all datasets when the cardinality of the dataset varies. The existing algorithm MR-BNL [9] is used as a reference. We vary the cardinality from $1,000,000$ to $500,000,000$. The average execution time of SSQ for all datasets are shown in Figure 8. As expected, the running time of the algorithms increases as the cardinality of the datasets increases on different distributions. SSQ is always better than



(a) anti-correlated (b) independent (c) correlated

Figure 8: Varying the Cardinality for skyline processing

MR-BNL because it can remove more non-skyline points using the grid-based pruning strategy. Furthermore, this trend becomes more evident as the size of the dataset increases because the pruning strategy has a greater effect for larger-scale datasets. As mentioned above, in Table 3, the first two parts of the total time are the preprocessing time of the algorithms. Because the preprocessing phase requires considerably more time compared with querying, particularly when the data size is small. The first two steps of SSQ with the SQM-filtering are the same as for SSQ with the ε-filtering. The difference is that the former can filter out some points by thresholds during the final query step. Compared to the SSQ algorithm with the ε-filtering, the execution time of SSQ with the SQM-filtering does not exhibit a significant improvement, but it can return the subset of subspace skyline points. By threshold filtering, the SSQ can reduce the cost of the query in the final step. Notably, as shown in Figure 8 (a), for the anti-correlated dataset, the effect is more obvious than for the other distributions because the SSQ algorithm can filter out more data points using the thresholds.

From the results in Figure 8, we observe that the algorithms with the anti-correlated datasets can require more execution time than those of the independent datasets and the correlated datasets. Because the number of skyline points in the anti-correlated datasets is generally larger than those in the other

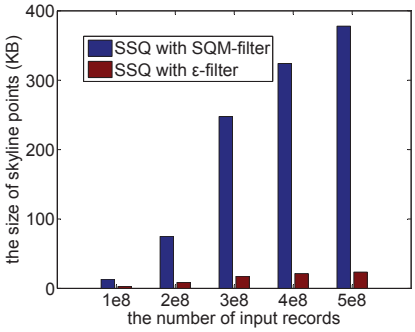
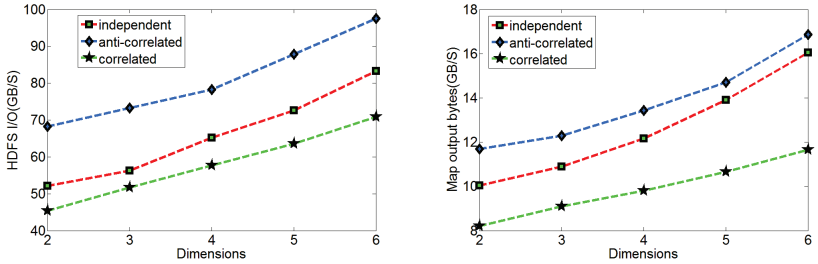


Figure 9: The size of result sets



(a) HDFS I/O cost

(b) Map output bytes

Figure 10: I/O cost of different data distributions

datasets. The performance of the pruning strategy is the worst for the anti-correlated dataset.

In the following set of experiments, we examine the size of the result set using the SSQ algorithm with the ε-filtering and compare it with that obtained using the SSQ algorithm with the SQM-filtering. We use the independent synthetic datasets to analyze the two filtering methods: SQM-filtering andε-filtering. We vary the number of input points from $100,000,000$ to $500,000,000$, and the filtering range of each threshold can be set to 5. As shown in Figure 9 shown, the ε-filtering method can return a smaller subset of skyline points compared with the SQM-filtering method.

We fix the cardinality of the synthetic datasets to $5e8$. We measure the cost

on datasets of three different distributions with the dimension varying from 2 to 6. The HDFS I/O cost includes read/write bytes upon HDFS (Hadoop Distributed File System) which is a fault-tolerant file system designed to run on commodity hardware. Figure 10 (a) shows the HDFS I/O costs of different distributions. With respect to the HDFS I/O cost, the cost for the anti-correlated distribution greatly exceeds those of the other distributions because it must read and write more records from the HDFS. The correlated dataset is exactly the opposite by entailing the fewest I/Os. Figure 10 (b) plots the map output bytes during pruning as the dimension increases from 2 to 6. The map output bytes include the number of bytes of uncompressed output produced by all of the maps in the job. As d becomes larger, the map output bytes continuously increase. The correlated dataset requires the least I/O cost. Furthermore, the output bytes of the anti-correlated distribution increase faster with increasing d. For the independent dataset, its curve lies between the those of the other two distributions.

7. CONCLUSION

In this paper, we present a system model for supporting subspace skyline query in mobile distributed environments. The subspace skyline query algorithm can report the meaningful subset of skyline points from the full set of skyline points. To reduce network communication and to shorten response time, we propose a pruning strategy for reducing the amount of data prior to computing the skyline points. To support subspace skyline query, we construct a global index according to value of each dimension using MapReduce. During the processing of skyline query, we extract the attributes of interest from the entire index. Finally, we eliminate some tuples that are worse in the important attributes using the SQM-filtering method or ε-filtering method. Our scheme alleviates the problem of there being too many skyline points to perform a manual evaluation. The experimental results indicate that the subspace skyline algorithm is highly efficient.

Acknowledgment

This work is supported by the National Nature Science Foundation of China (61173165, 61370199, 61300187, 61370198 61402069 and U1433124), JSPS KAKENHI Grant Number 26730056, JSPS A3 Foresight Program, the Fundamental Research Funds for the Central Universities (3132014325 and 3132013335), the General Project of Liaoning Provincial Department of Education Science Research (L2015092, L2014492, L2014283 and L2014191), the Prospective Research Project on Future Networks from Jiangsu Future Networks Innovation Institute.

References

[1] K. Hose, A. Vlachou, A survey of skyline processing in highly distributed environments, The VLDB Journal (2012) 359–384.

[2] S. Wang, B. C. Ooi, A. K. H. Tung, Efficient skyline query processing on peer-to-peer networks, in: IEEE International Conference on Data Engineering (ICDE), 2007, pp. 1126–1135.

[3] S. Wang, Q. H. Vu, B. C. Ooi, A. K. H. Tung, L. Xu, Skyframe: a framework for skyline query processing in peer-to-peer systems, VLDB (2009) 345–362.

[4] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, A. E. Abbadi, Parallelizing skyline queries for scalable distribution, in: EDBT06, 2006, pp. 112–130.

[5] J. Pei, W. Jin, M. Ester, Y. Tao, Catching the best views of skyline: A semantic approach based on decisive subspaces, in: VLDB, 2005, pp. 253–264.

[6] Y. Tao, Subsky: Efficient computation of skylines in subspaces, in: IEEE International Conference on Data Engineering(ICDE), 2006, p. 65.

[7] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM (2008) 107–113.

[8] L. Ding, G. Wang, J. Xin, X. S. Wang, S. Huang, R. Zhang, Commapreduce: An improvement of mapreduce with lightweight communication mechanisms, Data Knowl. Eng. (2013) 224–247.

[9] B. Zhang, S. Zhou, J. Guan, Adapting skyline computation to the mapreduce framework: Algorithms and experiments, in: DASFAA, 2011, pp. 403–414.

[10] L. Chen, K. Hwang, J. Wu, Mapreduce skyline query processing with a new angular partitioning approach, 2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (2012) 2262–2270.

[11] Y. Park, J. Min, K. Shim, Parallel computation of skyline and reverse skyline queries using mapreduce, PVLDB 6 (14) (2013) 2002–2013.

[12] C. yong Chan, H. V. Jagadish, K. lee Tan, A. K. H. Tung, Z. Z. S. O. Computing, Finding k-dominant skylines in high dimensional space, in: SIGMOD, 2006, pp. 503–514.

[13] S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: IEEE International Conference on Data Engineering(ICDE), 2001, pp. 421–430.

[14] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, Skyline with presorting, in: IEEE International Conference on Data Engineering(ICDE), 2002, pp. 717–719.

[15] D. Kossmann, F. Ramsak, S. Rost, Shooting stars in the sky: an online algorithm for skyline queries, in: VLDB, 2002, pp. 275–286.

[16] D. Papadias, Y. Tao, G. Fu, B. Seeger, An optimal and progressive algorithm for skyline queries., in: SIGMOD, 2003, pp. 467–478.

[17] W.-T. Balke, U. Gntzer, J. X. Zheng, Efficient distributed skylining for web information systems, in: EDBT, 2004, pp. 256–273.

[18] E. Lo, K. Y. Yip, K.-I. Lin, D. W. Cheung, Progressive skylining over web-accessible databases, Data Knowl. Eng. (2006) 122–147.

[19] G. Valkanas, A. N. Papadopoulos, Efficient and adaptive distributed skyline computation, in: SSDBM, 2010, pp. 24–41.

[20] A. Vlachou, C. Doulkeridis, Y. Kotidis, Angle-based space partitioning for efficient parallel skyline computation, in: SIGMOD, 2008, pp. 227–238.

[21] H. Köhler, J. Yang, X. Zhou, Efficient parallel skyline processing using hyperplane projections, in: SIGMOD, 2011, pp. 85–96.

[22] X. Han, J. Li, D. Yang, J. Wang, Efficient skyline computation on big data, IEEE Transactions on Knowledge and Data Engineering (2013) 2521–2535.

[23] H. Li, M. Dong, X. Liao, H. Jin, Deduplication-based energy efficient storage system in cloud environment, Comput. J. 58 (6) (2015) 1373–1383.

[24] J. a. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, K. Nørvåg, Agids: A grid-based strategy for distributed skyline query processing, in: Proceedings of the 2Nd International Conference on Data Management in Grid and Peer-to-Peer Systems, Globe '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 12–23.

[25] J. Wu, M. Dong, K. Ota, Z. Zhou, B. Duan, Towards fault-tolerant fine-grained data access control for smart grid, Wireless Personal Communications 75 (3) (2014) 1787–1808.

[26] C. Ji, Z. Li, W. Qu, Y. Xu, Y. Li, Scalable nearest neighbor query processing based on inverted grid index, J. Netw. Comput. Appl. 44 (2014) 172–182.

[27] O. Wolfson, Y. Zheng, S. Ma, T-share: A large-scale dynamic taxi ridesharing service, in: Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013), 2013, pp. 410–421.