PAPER Special Section on Data Engineering and Information Management

k-Dominant Skyline Query Computation in MapReduce Environment

Md. Anisuzzaman SIDDIQUE[†a)], **Hao TIAN**[†], ***Nonmembers***, ***and*** **Yasuhiko MORIMOTO**[†], ***Member***

SUMMARY Filtering uninteresting data is important to utilize "big data". Skyline query is popular technique to filter uninteresting data, in which it selects a set of objects that are not dominated by another from a given large database. However, a skyline query often retrieves too many objects to analyze intensively especially for high-dimensional dataset. To solve the problem, *k*-dominant skyline queries have been introduced. The size of databases sometimes become too large to compute in a centralized environment. Conventional algorithms for computing *k*-dominant skyline queries are not well suited for parallel and distributed environments, such as the MapReduce framework. In this paper, we consider an efficient parallel algorithm to process *k*-dominant skyline query in MapReduce framework. Extensive experiments demonstrate the scalability of proposed algorithm for synthetic big datasets under different settings of data distribution, dimensionality, and cardinality.

key words: *skyline query, k-dominant skyline query, MapReduce, big data*

1. Introduction

Filtering uninteresting data is an important step to utilize "big data". Moreover, to select representative distinctive objects in a database is important to understand the data in an early stage of knowledge discovery process. Skyline query is one of popular techniques for such data processing.

Let DS be an m-dimensional database. An object O is said to dominate another object O' if O is not worse than O' in any of the m dimensions and O is better than O' in at least one of the m dimensions. A skyline query retrieves a set of objects, each of which is not dominated by another object. Consider a symbolic skyline dataset with two attributes a_1 and a_2 as shown in Fig. 1. Without loss of generality, we assume smaller value is better in each dimension. Skyline query retrieves $\{o_3, o_4, o_5, o_6\}$ (see Figure 1 (b)).

Since the notion of the skyline operator [3] was introduced by Borzsonyi in 2001, it has attracted considerable attention due to its broad applications including product or restaurant recommendations [11], review evaluations with user ratings [10], querying wireless sensor networks [21], and graph analysis [24]. A number of efficient algorithms for computing skyline objects have been reported in the literature [5], [9], [15], [22].

One of known weakness of the skyline query is that it can not control the size of retrieved objects. It retrieves too many objects especially for high-dimensional databases. As a result, a user may be overwhelmed as s/he may have to examine numerous skyline objects manually to examine.

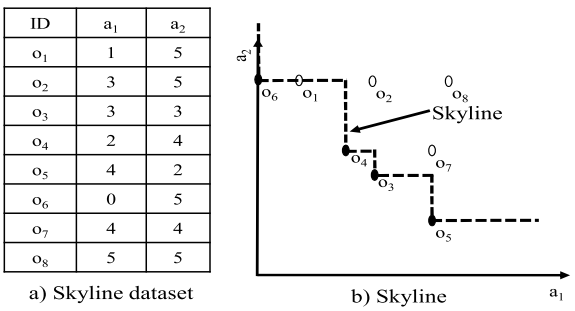


Fig. 1 Skyline example.

Table 1 *k*-dominant skyline dataset.

Object	a_1	a_2	a_3	a_4	a_5	a_6
O_1	4	1	5	3	1	4
O_2	8	2	2	1	7	5
O_3	6	6	9	7	1	9
O_4	2	8	1	7	2	2
O_5	8	9	6	7	5	6
O_6	3	3	8	2	4	5
O_7	1	5	9	2	8	6
O_8	5	2	2	4	7	9

To solve the weakness, Chan et al. considered *k*-dominant skyline query [4]. They relaxed the definition of "dominated" so that an object is more likely to be dominated by another object according to the parameter "*k*". Given an m-dimensional dataset DS, an object O is said to k-dominates another object O' if there are k ($k \leq m$) dimensions in which O is better than or equal to O'. A k-dominant skyline object is an object that is not k-dominated by any other object.

Assume another symbolic dataset for illustrating *k*-dominant skyline as listed in Table 1. In the table, each object has six dimensions from a_1 to a_6. Skyline query for this database returns six objects $\{O_1, O_2, O_4, O_6, O_7, O_8\}$ out of eight objects. Naturally, users want to eliminate less important objects from the result set. The k-dominant skyline query can control the selectivity by changing k. Consider the case where $k = 5$. Object "O_1" 5-dominates object "O_8" because all attributes of "O_1" except a_3 are equal or better than those of "O_8". Similarly, "O_7" is 5-dominated by "O_2". Others are not 5-dominated. Therefore, $\{O_1, O_2, O_4, O_6\}$ are 5-dominant skyline objects. If $k = 4$, object "O_1" 4-dominates "O_2" and "O_6". Therefore, $\{O_1, O_4\}$ are 4-dominant sky-

Manuscript received July 28, 2014.
Manuscript revised November 28, 2014.
Manuscript publicized January 21, 2015.

[†]The authors are with Hiroshima University, Higashi-Hiroshima-shi, 739–8521 Japan.
a) E-mail: siddique@hiroshima-u.ac.jp

DOI: 10.1587/transinf.2014DAP0010

line objects. If we decrease the value of k by one, "O_1" 3-dominates "O_4" and "O_1" is 3-dominated by "O_4". Therefore, 3-dominant skyline query retrieves empty set.

The size of databases sometimes becomes too large to compute in a centralized environment. Conventional algorithms for computing k-dominant skyline queries are not well suited for parallel and distributed environments, such as MapReduce framework. In this paper, we consider an efficient parallel algorithm to process k-dominant skyline queries in MapReduce framework. We utilize an spatial index structure, called object-based bound tree to split the data space so that we can compute sub-processes of the k-dominant skyline query on each subspace independently. Extensive performance study shows that our proposed algorithm is scalable to handle "big data".

The rest of this paper is organized as follows: Sect. 2 reviews related work. Section 3 presents the notions and properties of k-dominant skyline computation. We provide detailed examples and analysis of k-dominant skyline algorithm in Sect. 4. We experimentally evaluate the proposed algorithm in Sect. 5 under a variety of settings. Finally, Sect. 6 concludes the paper.

2. Related Work

Our work is motivated by previous studies of skyline query processing as well as MapReduce based query processing.

2.1 Skyline and k-Dominant Query Processing

Borzsonyi et al. first introduced the skyline operator over large databases and proposed three algorithms: *Block Nested-Loops* (BNL), *Divide-and-Conquer* (D&C), and B-tree-based schemes [3]. BNL compares each object of the database with every other object, and reports it as a result only if any other object does not dominate it. A window W is allocated in main memory, and the input relation is sequentially scanned. In this way, a block of skyline objects is produced in every iteration. In case the window saturates, a temporary file is used to store objects that cannot be placed in W. This file is used as the input to the next pass. D&C divides the dataset into several partitions such that each partition can fit into memory. Skyline objects for each individual partition are then computed by a main-memory skyline algorithm. The final skyline is obtained by merging the skyline objects for each partition. Chomicki et al. improved BNL by presorting, they proposed *Sort-Filter-Skyline* (SFS) as a variant of BNL [5]. Among index-based methods, Tan et al. proposed two progressive skyline computing methods Bitmap and Index [17]. In the Bitmap approach, every dimension value of an object is represented by a few bits. By applying bit-wise *AND* operation on these vectors, a given object can be checked if it is in the skyline without referring to other objects. The index method organizes a set of m-dimensional objects into m lists such that an object O is assigned to list i if and only if its value at attribute i is the best among all attributes of O. Each list is indexed by a B-tree, and the skyline is computed by scanning the B-tree until an object that dominates the remaining entries in the B-trees is found. The current most efficient method is *Branch-and-Bound Skyline* (BBS), proposed by Papadias et al., which is a progressive algorithm based on the *best-first nearest neighbor* (BF-NN) algorithm [15]. Instead of searching for nearest neighbor repeatedly, it directly prunes using the R*-tree structure.

Chan et al. introduce k-dominant skyline query [4]. They proposed three algorithms, namely, *One-Scan Algorithm* (OSA), *Two-Scan Algorithm* (TSA), and *Sorted Retrieval Algorithm* (SRA). OSA uses the property that a k-dominant skyline object cannot be worse than any skyline object on more than k dimensions. This algorithm maintains the skyline objects in a buffer during the scan of the dataset and uses them to prune away objects that are k-dominated. TSA retrieves a candidate set of dominant skyline objects in the first scan by comparing every object with a set of candidates. The second scan verifies whether these objects are truly dominant skyline objects or not. This method turns out to be much more efficient than the one-scan method. A theoretical analysis is provided to show the reason for its superiority. The third algorithm, SRA is motivated by the rank aggregation algorithm proposed by Fagin et al., which presorts data objects separately according to each dimension and then merges these ranked lists [7].

Another study on computing k-dominant skyline is k-ZSearch proposed by Lee et al. [12]. They introduced a concept called filter-and-reexamine approach. In the filtering phase, it remove all k-dominated objects and retain possible skyline candidates, which may contain false hits. In the reexamination phase, all candidates are reexamined to eliminate false hits.

Recently, more aspects of skyline computation have been explored. Lin et al. proposed n-of-N skyline query to support online query on data streams, i.e., to find the skyline of the set composed of the most recent n elements. In the cases where the datasets are very large and stored distributedly, it is impossible to handle them in a centralized fashion [13]. Balke et al. first mined skyline in a distributed environment by partitioning the data vertically [1]. Vlachou et al. introduce the concept of extended skyline set, which contains all data elements that are necessary to answer a skyline query in any arbitrary subspace [20]. Tao et al. discuss skyline queries in arbitrary subspaces [18]. More skyline variants such as dynamic skyline [14] and reverse skyline [6] operators also have recently attracted considerable attention.

2.2 MapReduce Based Query Processing

In order to handle "big data", the MapReduce [2], [8], [19] framework has recently attracted a lot of attentions. MapReduce is a programming model that allows easy development of scalable parallel applications to process big data on large clusters of commodity machines. Ideally, a MapReduce system should achieve a high degree of load balancing among the participating machines and minimize the space uses,

CPU and I/O time, and network transfer at each machine.

In [23], the authors discussed three skyline algorithms adapted to the MapReduce framework. *MapReduce-based Block Nested Loop* (MR-BNL) first partitions the data space into two dimensional subspaces base on the medians of each dimension and computes the local skyline for each subspace using BNL algorithm. Then, all the local skylines are merged into one machine to compute global skyline. *MapReduce-based Sort-Filter-Skyline* (MR-SFS) modifies MR-BNL by achieving presorting. *MapReduce-based Bitmap* (MR-Bitmap) builds the bitmap structure to examine each object in parallel. Both MR-BNL and MR-SFS concentrate on one machine to compute the global skyline. They suffer from dimensional curse because the size of local skyline will be much larger when dimensionality is large. MR-Bitmap requires a large amount of disk space to store bitmap structure if the bitmap structure cannot fit in a main memory. So MR-Bitmap has to spend the largest I/O cost.

Another parallel skyline algorithm based on MapReduce, namely SKY-MR, is proposed in [16]. The idea is based on a sky-quadtree to subdivide the data space recursively into sub-regions. The local skyline objects of each region are computed independently and then checked with objects from other regions which may dominate objects in this region. Partitioning based on sky-quadtree leads to unbalance in global skyline computation. The servers corresponding to the regions which are closer to the max corner of data space receive larger data to check than other servers, even though they do not contribute to the global skyline at all.

In this paper, we consider MapReduce based *k*-dominant skyline query. To the best of the authors' knowledge, there is no MapReduce based algorithm for *k*-dominant skyline query.

3. Preliminaries

In this section, we present some definitions and basic properties of our algorithm.

3.1 Skyline and *k*-Dominant Skyline

Assume we have a database DS with m-attributes $\{a_1, a_2, \cdots, a_m\}$. The database is distributed into n datasets $\{DS_1, DS_2, \cdots, DS_n\}$ on different locations. We use $O_{i,j}.a_k$ to denote the k-th attribute's value of object $O_{i,j}$ where i denotes datasets ID and j denotes object ID in the corresponding dataset DS_i.

For simplicity, assume that the symbolic k-dominant skyline dataset shown in Table 1 is splitted into two smaller datasets, DS_1 and DS_2, and distributed in different locations as shown in Table 2 and 3, respectively. In Table 2 and 3, the first suffix of object ID represents data source ID number and the second one represents object ID in the corresponding data source. For example, $O_{2,2}$ is an object of DS_2 and its ID in DS_2 is "2".

Table 2 Distributed k-dominant skyline dataset DS_1.

ID	a_1	a_2	a_3	a_4	a_5	a_6
$O_{1,1}$	4	1	5	3	1	4
$O_{1,2}$	8	2	2	1	7	5
$O_{1,3}$	6	6	9	7	1	9
$O_{1,4}$	2	8	1	7	2	2

Table 3 Distributed k-dominant skyline dataset DS_2.

ID	a_1	a_2	a_3	a_4	a_5	a_6
$O_{2,1}$	8	9	6	7	5	6
$O_{2,2}$	3	3	8	2	4	5
$O_{2,3}$	1	5	9	2	8	6
$O_{2,4}$	5	2	2	4	7	9

Definition *Dominate*: For objects $O_{i,j}$ and $O_{x,y}$ (where if i = x then j ≠ y or if j = y then i ≠ x), an object $O_{i,j}$ is said to *dominate* another object $O_{x,y}$, denoted by $O_{i,j} \leq O_{x,y}$, if $O_{i,j}.a_s \leq O_{x,y}.a_s$ for all attributes ($s = 1, \cdots, m$) and $O_{i,j}.a_s < O_{x,y}.a_s$ for at least one attribute s in the m attributes. We call such $O_{i,j}$ as *dominant* object and such $O_{x,y}$ as *dominated* object between $O_{i,j}$ and $O_{x,y}$. If $O_{i,j}$ dominates $O_{x,y}$, then $O_{i,j}$ is more preferable than $O_{x,y}$.

Definition *Skyline*: An object $O \in DS$ is in skyline of DS (i.e., a skyline object in DS) if O is not dominated by any other object in DS. The skyline of DS, denoted by $Sky(DS)$, is the set of skyline objects in DS. For distributed skyline dataset DS (shown in Table 2 and 3), object $O_{1,1}$ dominates $O_{1,3}$ and $O_{2,1}$. The other objects, i.e., $O_{1,1}, O_{1,2}, O_{1,4}, O_{2,2}, O_{2,3}$, and $O_{2,4}$ are not dominated by another object in DS. Thus, skyline query on Table 2 and 3 will retrieve $Sky(DS) = \{O_{1,1}, O_{1,2}, O_{1,4}, O_{2,2}, O_{2,3}, O_{2,4}\}$.

Definition *k-Dominate*: For objects $O_{i,j}$ and $O_{x,y}$ (where if i = x then j ≠ y or if j = y then i ≠ x), an object $O_{i,j}$ is said to *k-dominate* another object $O_{x,y}$, denoted by $O_{i,j} \leq_k O_{x,y}$, if $O_{i,j}.a_s \leq O_{x,y}.a_s$ for $k (k \leq m)$ attributes among the m attributes and $O_{i,j}.a_s < O_{x,y}.a_s$ for at least one attribute. We call such $O_{i,j}$ as *k-dominant* object and such $O_{x,y}$ as *k-dominated* object between $O_{i,j}$ and $O_{x,y}$. Similar to skyline object, if $O_{i,j}$ k-dominates $O_{x,y}$, then $O_{i,j}$ is more preferable than $O_{x,y}$.

Definition *k-dominant Skyline*: An object $O \in DS$ is in k-dominant skyline of DS if O is not k-dominated by another object in DS. The k-dominant skyline of DS, denoted by $Sky_k(DS)$, is the set of k-dominant skyline objects in DS. In DS (shown in Table 2 and 3), $\{O_{1,1}, O_{1,2}, O_{1,4}, O_{2,2}, O_{2,3}, O_{2,4}\}$ are not dominated by another object. Thus, $Sky_6(DS)$, which is $Sky(DS)$, will retrieve $\{O_{1,1}, O_{1,2}, O_{1,4}, O_{2,2}, O_{2,3}, O_{2,4}\}$. Now, if we consider 5-dominant skyline, $O_{1,1}$ 5-dominates $O_{2,4}$ because $O_{1,1}$ is equal or better in a_1, a_2, a_4, a_5, and a_6. Similarly, $O_{1,2}$ 5-dominates $O_{2,3}$. The other objects are not 5-dominated. Thus, $Sky_5(DS) = \{O_{1,1}, O_{1,2}, O_{1,4}, O_{2,2}\}$. If k = 4, $O_{1,1}$ 4-dominates $O_{1,2}$, $O_{2,2}$. Thus, $Sky_4(DS) = \{O_{1,1}, O_{1,4}\}$. Since $O_{1,1}$ 3-dominates $O_{1,4}$ and $O_{1,1}$ is 3-dominated by $O_{1,4}$, $Sky_3(DS)$ becomes an empty set, since all objects are 3-dominated by each other.

3.2 Hadoop MapReduce

MapReduce was proposed by Google as a parallel and distributed programming framework for large-scale data processing. In the MapReduce framework, data are represented as key/value pairs and computation is distributed across a shared nothing cluster of commodity machines. A job to be performed using the MapReduce framework mainly refers to two user-defined functions, called Map and Reduce:

$$Map(k_1, v_1) \rightarrow list(k_2, v_2) \quad (1)$$

$$Reduce(k_2, list(v_2)) \rightarrow list(v_3) \quad (2)$$

The Map function (also called Mapper) processes on each (key, value) pair of input data, and produces new (key, value) pairs. The intermediate (key, value) pairs are then grouped and sorted associated with the same intermediate key. The Reduce function (also called Reducer) takes a key and a list of values for that key, applies the processing logic, and generates the final result as a list of values.

Hadoop is an open source implementation of the MapReduce framework. The Hadoop MapReduce framework is designed to allow users to program a MapReduce job only by defining the map and reduce functions. A MapReduce job usually splits the input dataset into equal-sized segments of typically 64 MB per segment.

4. *k*-Dominant Skyline Processing

In this section, we present our approach to process *k*-dominant skyline queries in MapReduce. Proposed algorithm consists of the following three stages, which are (1) object-based bound tree construction, (2) MapReduce for candidate selection, and (3) MapReduce for final skyline.

4.1 Object-Based Bound Tree Construction

As the first stage, we choose an adequate size samples from a database and then construct a tree called Object-based bound tree (OB-Tree), from the samples. Using *Euclidean* distance order we find an object closest to the origin. We use the closest object as root pivot $O^v = (O^v.s_1, O^v.s_2, \ldots, O^v.s_m)$. Next, by using the pivot, we divide the m-dimensional space S^m into m non-disjoint subspaces $S^m_1, S^m_2, \ldots, S^m_m$, where $S^m_i (i = 1, \ldots, m)$ is the lower subspace divided by hyperplane $s_i = O^v.s_i$. If there are subspaces that contain more samples than a user-specified value, we recursively divide such subspaces.

Assume we have a dataset shown in Fig. 2. The m-ary tree in the figure is an OB-Tree constructed from the eight sample objects. In this sample, o_3 is the root pivot. We use $o_3 = (3, 3)$ to divide the 2D space into S_1 ($a_1 \leq 3$) and S_2 ($a_2 \leq 3$). S_1 contains five objects $\{o_1, o_2, o_3, o_4, o_6\}$. S_2 contains two objects $\{o_3, o_5\}$. If the user-specified value for sample size is 3, we recursively divide S_1 to $S_{1.1}$ and $S_{1.2}$ by using o_4. Finally, we have three subspaces $S_{1.1}$ ($a_1 \leq 2$), $S_{1.2}$ ($a_1 \leq 3$ and $a_2 \leq 4$), and S_2 ($a_2 \leq 3$).

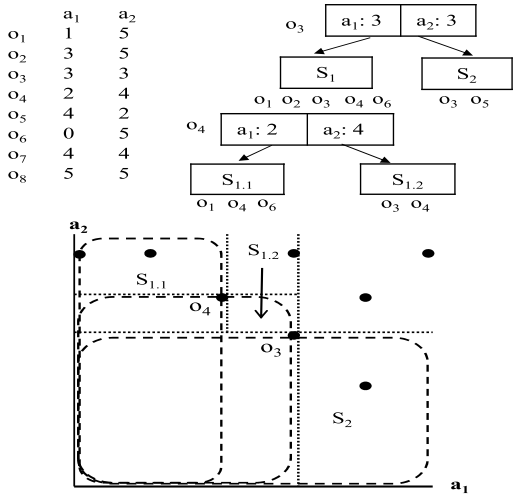


Fig. 2 2D sample and OB-Tree.

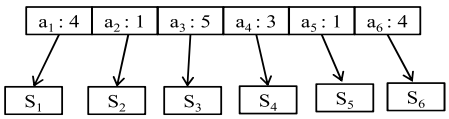


Fig. 3 OB-Tree of *k*-dominant skyline dataset.

4.2 MapReduce for Candidate Selection

We filter and dispatch each object in the database by using OB-Tree. Assume we have OB-Tree as in Fig. 3 which is constructed from Table 1. Since the symbolic dataset is small and relatively high dimensional, the depth of the OB-Tree is one and we assume that $O_{1,1} = (4, 1, 5, 3, 1, 4)$ is the root pivot that splits the space into six non-disjoint subspaces, say $S_1, S_2, \ldots, S_6$.

We generate key-value pairs as the second column in Fig. 4. In the key-value pair, key is subspace and value is a triplet of an object. For example, object $O_{1,2}$ is contained in two subspaces S_3 and S_4. For each subspace that contains $O_{1,2}$, we generate key-value pair for $O_{1,2}$ with $(O_{1,2}, 2, +)$, where $O_{1,2}$ is `id` of an object, "2" is the number of generated pairs for the object, and "+" is a sign that shows whether the object can be a candidate of *k*-dominant skyline. In this example, we compute each sign with $k = 5$.

Let $S(O)$ and $|S(O)|$ be subspaces that contain O and the number of the subspaces, respectively. We can apply following theorems.

Theorem 1: If O is a skyline object (m-dominant skyline object), $|S(O)| \geq 1$.

Proof: If O is a skyline object, O can not be dominated by any pivot in an OB-Tree. Therefore, O must be included in at least one subspace. □

According to Theorem 1, we can conclude that all *k*-dominant skyline objects must be in at least one subspace, since a *k*-dominant skyline object ($k \leq m$) must be a skyline

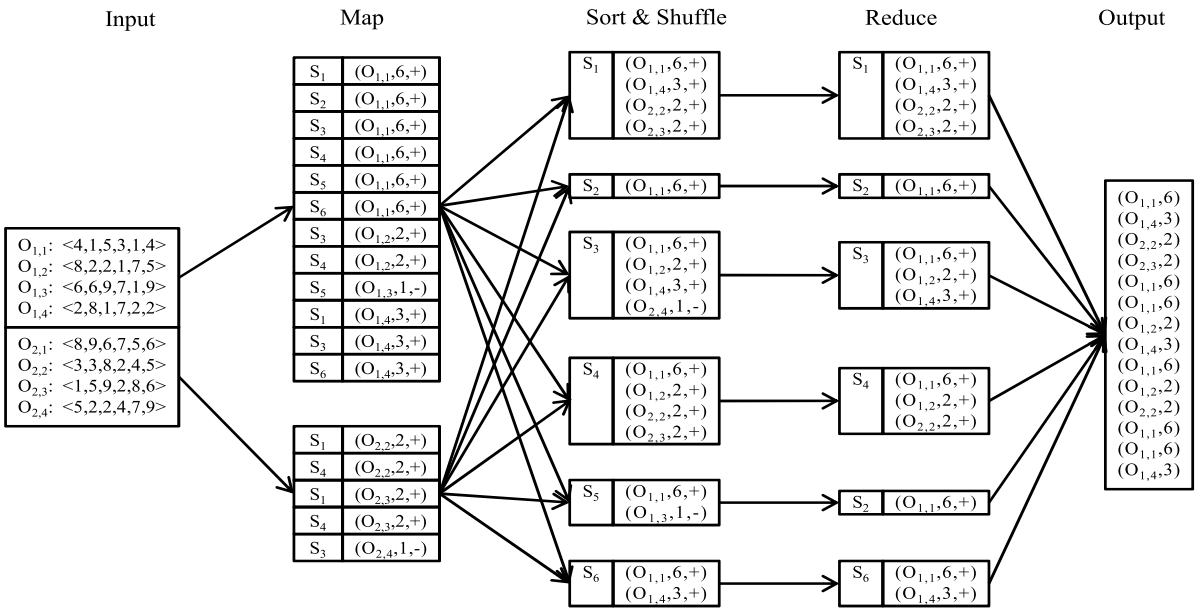


Fig. 4 MapReduce data flow of candidate selection.

object. This also means that all k-dominant skyline objects must contained in the key-value pairs.

Theorem 2: If O in the m-dimensional space is not k-dominated by a pivot, then $|S(O)| \geq m - k + 1$. In other words, if $|S(O)| < m-k+1$, then the object O is k-dominated by a pivot and cannot be a candidate of k-dominant skyline object.

Proof: Each internal node of OB-Tree has m child-nodes. If O is contained only in $m - k$ subspaces (child-nodes) of a pivot (an internal node), the pivot is strictly better than O in k dimensions. If the number of subspaces that contains O, $|S(O)|$, is at most $m-k$, one of pivots in OB-Tree k-dominates O. □

By using Theorem 2, we compute the sign of each key-value pair. Since $|S(O_{1,3})| < 2(m - k + 1 = 6 - 5 + 1)$ and $|S(O_{2,4})| < 2$, objects $O_{1,3}$ and $O_{2,4}$ cannot be 5-dominate skyline. Therefore, the sign of the triplet of those objects is "–".

Each mapper dispatch the key-value pairs to reducers based on the key as in Fig. 4. Each reducer then examines the "local" k-dominance for each "+" object and outputs local k-dominant skyline. For example, reducer S_4 receives four + objects (as in the 3rd column). It examines each of the four objects whether it is locally 5-dominated by another object in the four objects. The reducer finds that $O_{2,3}$ is 5-dominated by $O_{1,2}$ and removes $O_{2,3}$ (as in the 4th column).

From the union of each reducer's output, we can generate key-value pairs for the next MapReduce computation. Each pair consists of candidate object "id" and the number of subspaces that contain the object.

4.3 MapReduce for Final Skyline

The output of key-value pairs of the candidate selection stage become the input of the final skyline stage. In this stage, mappers dispatch key-value pairs based on the key (object "id" such as $O_{1,1}, O_{1,4}$, etc.). Each reducer counts the number of key-value pairs for each candidate object.

Let $|O|$ be the number of the key-value pairs for O in the input. The results of Fig. 4 are $|O_{1,1}| = 6$, $|O_{1,2}| = 2$, $|O_{1,4}| = 3$, $|O_{2,2}| = 2$, and $|O_{2,3}| = 1$. Notice that $|O_{2,3}| = 1 < |S(O_{2,3})| = 2$ (The number of key-value pairs for $O_{2,3}$ is smaller than the number of subspaces that contain $O_{2,3}$.), which implies that $O_{2,3}$ is 5-dominated by another object in at least one of subspaces in $S(O_{2,3})$.

Theorem 3: Given two objects O and O'. If O is not a pivot object and can k-dominate O', then $S(O) \cap S(O') \geq 1$.

Proof: Let us consider two objects, O and O' such that object O is not a pivot and it k-dominates O'. We first consider a case where OB-Tree has only one pivot object, which implies the depth of the tree is one. Assume that $S(O) \cap S(O') = \emptyset$. According to Theorem 2, at least $m-k+1$ subspaces contains O'. In other words, at most $k-1$ subspaces do not contain O'. Object O may be better than or equal to O' in all the corresponding $(k-1)$-dimensions. However, since O k-dominates O', O must be better than or equal to O' in at least k dimensions. This implies that O must be better than or equal to O' in at least one dimension that corresponds to the subspace containing O'. So, O must exist in at least one subspace that contains O'. This contradicts the assumption $S(O) \cap S(O') = \emptyset$, thus $S(O) \cap S(O') \geq 1$.

However, if the depth of OB-Tree is greater than one, which means that there exists more than one pivot objects. Then, a subspace splitted by another pivot (except the root pivot) contains both O and O'. By applying similar procedure we can prove that at least one child-subspace contains both objects. □

From Theorem 1 and Theorem 3, it is clear that if an

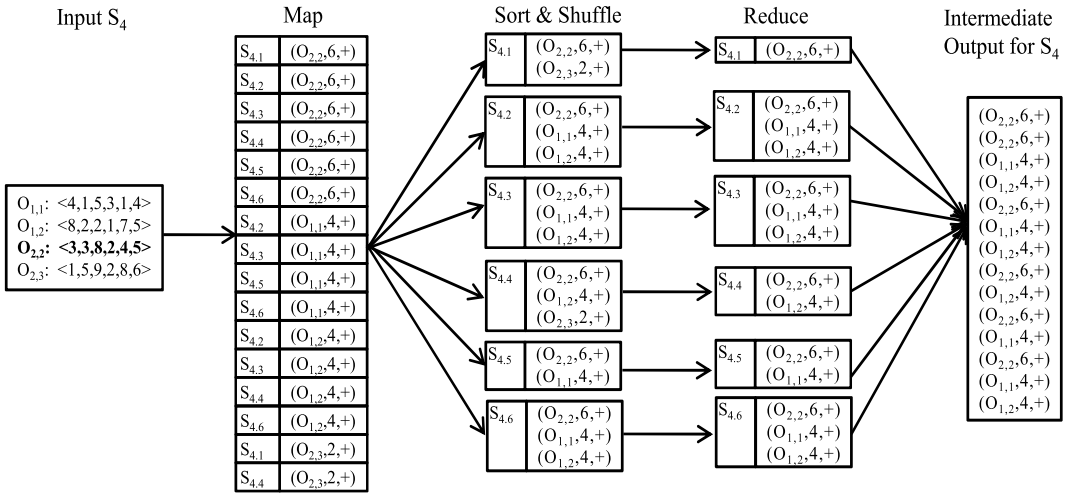


Fig. 5 Data flow of candidate selection in the 2nd level of OB-Tree.

object O has the property $|S(O)| \geq 1$ and is not k-dominated by any pivots as well as other objects in $S(O)$, O is a member of the global k-dominant skyline. Therefore, the 5-dominant skyline of Table 1 is $Sky_5(DS) = \{O_{1,1}, O_{1,2}, O_{1,4}, O_{2,2}\}$.

4.4 Procedures in the 2nd Level of OB-Tree

To make better understanding, we explain our candidate selection procedures in a case where the depth of OB-Tree is greater than one. Assume that the user specified value (the maximum number of objects in a subspace) is 3. Then, the subspaces that have more than three objects are split. In the example of Fig. 4, S_1, S_3, and S_4 among six child nodes of the root are further splitted. Here, we present the split of S_4 in Fig. 5. Using Euclidean distance order, we choose object $O_{2,2}$ as our new pivot because it is the closest from the origin. By applying the splitting procedure with $O_{2,2}$, S_4 is split into six subspaces, say $S_{4.1}, \cdots, S_{4.6}$. Each mapper sends the key-value pairs to reducers based on the key. Next, each reducer checks intermediate k-dominance for each "+" object and outputs intermediate k-dominant skyline. For example, reducer $S_{4.1}$ and $S_{4.4}$ receive 2 and 3 "+" objects, respectively and find that in both subspaces object $O_{2,2}$ becomes 5-dominant of object $O_{2,3}$. Finally, by computing the number of the key-value pairs for each object we get $Sky_5(S_4) = \{O_{2,2}, O_{1,1}, O_{1,2}\}$ as intermediate 5-dominant skyline result for S_4.

Note that the user specified value (the maximum number of objects in a subspace), i.e., 3, in this subsection is too small. We used this value just to explain the split procedure in the 2nd level of the OB-Tree.

5. Performance Evaluation

We set up a cluster of 4 commodity PCs in a high speed Gigabit networks, each of which has an Intel Core i7 3.4GHz CPU, 4GB memory and Windows 8.0 OS. The machines are connected with a Gbps LAN connection. We compile the source codes under JDK 1.6. We conduct a series of experiments with different data distributions, dimensionalities, and data cardinalities to evaluate the effectiveness and efficiency of our proposed methods. Each experiment is repeated five times and the average result is considered for performance evaluation. Three data distributions proposed in [3] are considered as follows:

Correlated: a correlated dataset represents an environment in which, objects are good in one dimension are also good in the other dimensions. In a correlated dataset, fairly few objects dominate many other objects.

Anti-Correlated: an anti-correlated dataset represents an environment in which, if an object has a small coordinate on some dimensions, it tends to have a large coordinate on at least another dimension.

Independent: for this type of dataset, all attribute values are generated independently using uniform distribution. Under this distribution, the total number of non-dominating objects is between that of the correlated and the anti-correlated datasets.

We implemented the proposed algorithm described in Sect. 4, denoted as **MR-DSKY**. In lack of techniques dealing directly with the problem of MapReduce based k-dominant skyline computation, we compare our method against TSA, which was the most efficient k-dominant skyline search proposed in [4]. To handle parallel computation in MapReduce environment, we adapt a variant of the **TSA** called **MR-TSA** (MapReduce based Two-Scan Algorithm). **MR-TSA** assigns map task to several workers to compute candidate skyline objects, then utilizes the algorithm **TSA**, to perform k-dominant skyline computation in one reduce worker. We set dimension $m = 10$, data cardinality = 1M, and $k = 8$ as the default values. Due to curse of dimensionality when m increases the number of pruned objects decreases. As a result, both **MR-TSA** and **MR-DSKY** become slower. Increasing value of k has also a similar effects on both algorithms.

However, in **MR-TSA** since the map workers need to

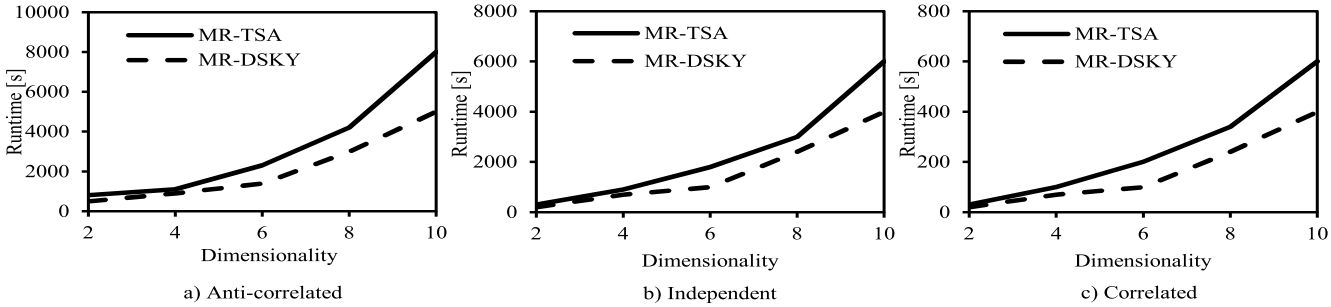


Fig. 7 Varying dimensionality.

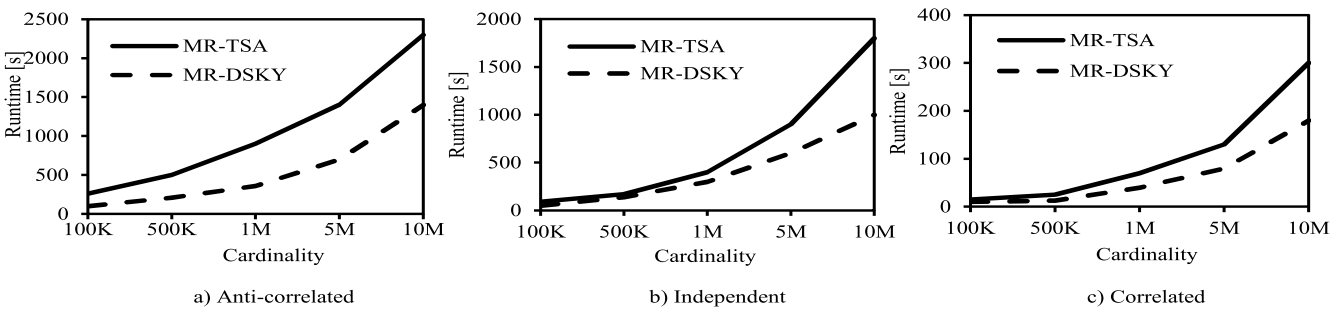


Fig. 8 Varying cardinality.

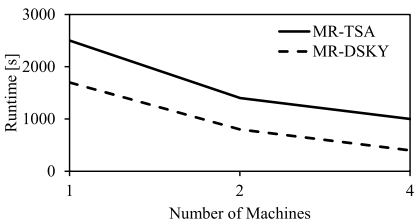


Fig. 6 Varying number of machines.

compute local skyline first and then local skyline objects of map workers are sent to the one reduce worker to compute *k*-dominant skyline. It generally take more execution overhead than **MR-DSKY**. This is because **MR-DSKY** do not need skyline computation for *k*-dominant candidate generation. In addition during the skyline computation procedure if the number of pruned objects decreases then the **MR-TSA** become inefficient. In **MR-TSA** huge number of pruned skyline objects are sent to the single reducer, as a result, it becomes inefficient due to high network costs. Decreasing value of *k* has also similar effect because it can not skip skyline computation.

5.1 Effect of Scalability

Figure 6 presents the runtime of the both algorithms by varying the number of machines. Our proposed algorithm **MR-DSKY** shows the best scalability since **MR-DSKY** effectively prunes data by partitioning with the OB-Tree and utilize pivot objects to prune unnecessary objects. As a result, **MR-DSKY** succeeded to avoid many unnecessary comparisons and had less computational overhead rather than **MR-TSA**.

5.2 Effect of Dimensionality

We study the effect of dimensionality on our MapReduce techniques. We fix the data cardinality to 1M, vary dimension m from 2 to 10, and for each case set *k* to (m-1). The runtime results for this experiment are shown in Fig. 7 (a), (b), and (c). The result shows that as the *k* increases the performance of MR-TSA become slower. It is interesting to note that there exist significant performance difference between both methods. However, **MR-DSKY** always outperform than **MR-TSA**.

5.3 Effect of Cardinality

For this experiment, we fix the data domensionality m to 10, *k* to 8, and vary dataset cardinality ranges from 100K to 10M. Figure 8 (a), (b), and (c) shows the performance on three data distributions. Both of the techniques are highly affected by data cardinality. If the data cardinality increases then the performances decreases. It shows that the performance of **MR-DSKY** is two times faster than **MR-TSA**.

6. Conclusion

This paper addresses a distributed computation of *k*-dominant skyline query in MapReduce environment. Proposed *k*-dominant skyline computation algorithm utilizes an object-based bound tree (OB-Tree) to split data space for parallel computation. It has been seen that by constructing OB-Tree, we can easily distribute necessary workload

among the map workers and compute the k-dominant skyline query result efficiently. Using synthetic datasets, we demonstrate the scalability of proposed method. Intensive experiments show the effectiveness and superiority against conventional methods.

It is worthy of being mentioned that this work can be expanded in a number of directions. First, from the perspective of parallel computing, how to compute k-dominant skyline from streaming dataset. Secondly, to design more efficient index based such as R-tree or B-tree based MapReduce algorithm are promising research topics.

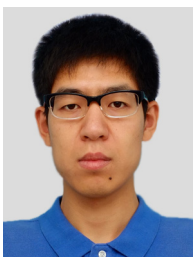
Acknowledgments

This work is supported by KAKENHI (23500180, 25.03040) Japan.

References

[1] W.-T. Balke, U. Güntzer, and J.-X. Zheng, "Efficient distributed skylining for web information systems," Proc. EDBT, pp.256–273, 2004.
[2] S. Blanas, J.M. Patel, V. Ercegovac, J. Rao, E.J. Shekita, and Y. Tian, "A comparison of join algorithms for log processing in mapreduce," Proc. SIGMOD, pp.975–986, 2010.
[3] S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator," Proc. ICDE, pp.421–430, 2001.
[4] C.Y. Chan, H.V. Jagadish, K.-L. Tan, A.K.H. Tung, and Z. Zhang, "Finding k-dominant skyline in high dimensional space," Proc. ACM SIGMOD, pp.503–514, 2006.
[5] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," Proc. ICDE, pp.717–719, 2003.
[6] E. Dellis and B. Seeger, "Efficient computation of reverse skyline queries," Proc. VLDB, pp.291–302, 2007.
[7] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," Proc. ACM PODS, pp.102–113, 2001.
[8] D. Jiang, A.K.H. Tung, and G. Chen, "Map-join-reduce: Toward scalable and efficient data analysis on large clusters," IEEE Trans. Knowl. Data Eng., vol.23, no.9, pp.1299–1311, 2011.
[9] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries," Proc. VLDB, pp.275–286, 2002.
[10] T. Lappas and D. Gunopulos, "Efficient confident search in large review corpora," Proc. PKDD Conference, pp.467–478, 2010.
[11] J. Lee, S. Hwang, Z. Nie, and J.-R. Wen, "Navigation system for product search," Proc. ICDE Conference, pp.1113–1116, 2010.
[12] K.C.K. Lee, B. Zheng, H. Li, and W.C. Lee, "Approaching the skyline in z order," Proc. VLDB, pp.279–290, 2007.
[13] X. Lin, Y. Yuan, W. Wang, and H. Lu, "Stabbing the sky: Efficient skyline computation over sliding windows," Proc. ICDE, pp.502–513, 2005.
[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," Proc. SIGMOD, pp.467–478, 2003.
[15] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," ACM Trans. Database Systems, vol.30, no.1, pp.41–82, 2005.
[16] Y. Park, J.-K. Min, and K. Shim, "Parallel computation of skyline and reverse skyline queries using mapreduce," Proc. VLDB, pp.2002–2013, 2013.
[17] K.-L. Tan, P.-K. Eng, and B.C. Ooi, "Efficient progressive skyline computation," Proc. VLDB, pp.301–310, 2001.
[18] Y. Tao, X. Xiao, and J. Pei, "Subsky: Efficient computation of skylines in subspaces," Proc. ICDE, p.65, 2006.
[19] R. Vernica, M.J. Carey, and C. Li, "Efficient parallel set-similarity joins using mapreduce," Proc. SIGMOD, pp.495–506, 2010.
[20] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis, "Skypeer: Efficient subspace skyline computation over distributed data," Proc. ICDE, pp.416–425, 2007.
[21] G. Wang, J. Xin, L. Chen, and Y. Liu, "Energy efficient reverse skyline query processing over wireless sensor networks," IEEE Trans. Knowl. Data Eng., vol.24, no.7, pp.1259–1275, 2012.
[22] T. Xia, D. Zhang, and Y. Tao, "On skylining with flexible dominance relation," Proc. ICDE, pp.1397–1399, 2008.
[23] B. Zhang, S. Zhou, and J. Guan, "Adapting skyline computation to the mapreduce framework algorithms and experiments," Proc. DASFAA, pp.403–414, 2011.
[24] L. Zou, L. Chen, M.T. Ozsu, and D. Zhao, "Dynamic skyline queries in large graphs," Proc. DASFAA Conference, pp.62–78, 2010.

Md. Anisuzzaman Siddique received his B.Sc., and M.Sc. degrees in Computer Science and Technology from University of Rajshahi (RU), Bangladesh in 2000 and 2002, respectively. He also received his D. Engg. degree from Hiroshima University in 2010. Since 2002-present he is a faculty member in RU. His research interests include skyline evaluation, data mining, and privacy preserving information retrieval.



Hao Tian received his B.E. degree in Software Engineering from Dalian University of Technology, China. Since 2012-present he is a Master student under the direction of Professor Yasuhiko Morimoto in Hiroshima University. His research interests include data management and distributed processing.

Yasuhiko Morimoto is an Associate Professor at Hiroshima University. He received his B.E., M.E., and Ph.D. degrees from Hiroshima University in 1989, 1991, and 2002, respectively. From 1991 to 2002, he had been with IBM Tokyo Research Laboratory where he worked for data mining project and multimedia database project. Since 2002, he has been with Hiroshima University. His current research interests include data mining, machine learning, geographic information system, and privacy preserving information retrieval.