Introduction

Given a d-dimensional data set, a point p dominates another point q if it is better than or equal to q in all dimensions and better than q in at least one dimension. A point is a skyline point if there does not exists any point that can dominate it. Skyline queries, which return skyline points, are useful in many decision making applications. Unfortunately, as the number of dimensions increases, the chance of one point dominating another point is very low. As such, the number of skyline points become too numerous to offer any interesting insights. To find more important and meaningful skyline points in high dimensional space, a concept , called k-dominant skyline had been proposed which relaxes the idea of dominance to k-dominance. A point p is said to k-dominate another point q if there are k (≤ d) dimensions in which p is better than or equal to q and is better in at least one of these k dimensions. A point that is not k-dominated by any other points is in the k-dominant skyline. k-dominant Skyline queries have been studied in centralized. However, the execution of k-dominant skyline queries on different (potentially overlapping) data fractions, in order to obtain the k-dominant skyline set of the entire dataset efficiently, has not received adequate attention. Here we study the computation of k-dominant skyline query in large-scale Distributed Environment such as peer to peer system in which every server site is connected to every other site. In such a setting, each server stores autonomously a fraction of the data(i.e. horizontal partitioning of data), thus all servers need to process the k-dominant skyline query.

The naïve solution for this problem will be to collect the data

To find the k-dominant skyline, we introduce a novel framework, called k-dom SkyAlgo, for processing distributed k-dominant skyline queries that returns the desired result.

Motivation

Why the k-dominant skyline have gained so much importance in the multi criteria decision making applications. For justifying this question , we will present some examples where , it is worth to compute the k-dominant skyline.

Example 1.

Suppose you are going to have a round trip around the world. In your trip you want to visit the popular places according to some criteria. We can clearly see that this system will have data to be distributed among the different sites. According to the preferences of the user we want to find some interesting locations or skyline points.

Example 2.

Suppose you are about to buy a product (say car) and suppose the database for storing the product’s information is distributed. Now, there could be a large number of attributes (like price ,color, mileage, green core) over which that product is being defined. There are too many cars at the website for the user to examine them all manually. So rather, You could select some or all of the attributes as your preference list. Computing the skyline over these cell phone features may remove a large number of cars whose features are “worse” than those in the skyline, hopefully leaving only a manageable number of candidates for manual evaluation.

The above examples illustrates that how useful it could be to compute the k-dominant skylines over a set of distributed databases.

The remaining of this paper is organized as follows: Section 2 overviews the related work. Then, we present the necessary preliminaries in Section 3 and the overview of k-dom Sky-Algo in Section 4. The results and evaluations in section 5.

2. Related Work

The basic idea of skyline by presented by Borzonyi et al, who proposed the very first algorithm to compute the skyline fom a given dataset. This approach was extended to a concept called k-dominant skyline computation by Chee-Yong Chan and his team. As the k-dominant skylines may be cyclic and non-transitive ,Thus the already existing algorithm can’t be applied to obtain the desired result. So They (Chee-Yong Chan and his team) proposed three new algorithms to get the k-dominant skylines. One scan algorithm basically sequentially scans the complete data set once and uses the property that each non –k dominant skyline will be dominated by at least one full skyline. Two scan algorithm firstly figures out a list of candidate set for the k-dominant skyline and then takes one more pass in the candidate set so as to remove the non-k dominant skyline. Sorted retrieval which is based on the aggregation based skyline.

The skyline computation in distributed environment was studied by João B. Rocha-Junior and his team. This paper was basically organized to get a execution plan so as to obtain a balance between the latency time and the network overhead.

We are basically combining the k-dominant skyline computation with the distributed environment. We are using the one scan algorithm to compute the k-dominant skyline at each site and as the execution plan computed by Rocha we are building a tree like structure to define the parallelism.

3.Preliminaries

Given a dataset D on a data space defined by a set of d dimensions {d1, ..., dd}, a point p ∈ D is represented as p={p1, ..., pd} where pi is the value on dimension di. Without loss of generality, we assume that ∀di : pi ≥ 0, and that smaller values are preferable.

The data set at any site Si is denoted by the MBR mi (li,ui)consisting of boundary of its data point.

We define the relations between the datasets of two different sites as:

* Site Si is said to fully dominate site Sj if ui>lj.
* Site Si is said to partially dominate the site Sj if ui>uj.
* If none of the above two cases occurs then sites are said to be incomparable.

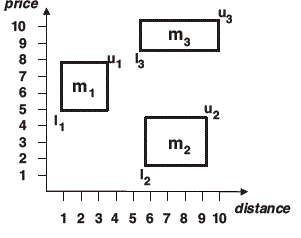


Fig 1.2 : Site Relation

In the figure 1.2, m1 fully dominate m3 and hence no point in m3 can be in the k-dominant skyline and hence m3 could be pruned off from the computation. M2 partially dominates m3 hence the points which lies strictly above the upper bound (u2) of M2 can be pruned off. Thus in general it would be beneficial to first compute the skylines in M2 then in M3. Hence , we make M2 as child of M3  during the tree construction.

The above mentioned relation between the different server sites forms the basis for our algorithm. Besides that we use one scan algorithm to find out the skylines at particular site’s data which is not pruned.

4. k-dom Sky-Algo

The basic steps involved in the algorithm are:

1. Collect the summarized information (i.e. MBR) of data of each server to the originator.
2. The originator the based on the relation between different server defines the parent child relation i.e. basically creates a tree like structure so as to execute in parallel.
3. Now, these parent child relations will be pushed to each server site.
4. After this tree formation phase, all the leaf nodes will start computing the k-dominant skyline in its own database and will push the results into the parent.
5. The parent node or server will compute its’ own k-dominant set and merge it with the result obtained from its’ children.
6. This process will keep on going till all are results have been accumulated to the originator.

The detailed algorithm is :

**4.1 k-dom Sky-Algo:**

* 1. for all sites m[i] 🡨 Compute\_MBR ( );

1. T 🡨 Tree\_Formation(m[i],sitedetails );
2. Push tree to each site.
3. At each site call k-domskyfinder( );

k-domskyfinder( ) algo:

1. input Query Q, Forest F={ T1,T2…..},filter ;
2. If F is empty
3. (local$\_$kdom,local\_nonk-domfullsky) <-- localoneScan(filter);
4. Else
5. localnonk-domfullsky= localoneScan (filter);
6. remoteFilter =localMBR.getUpperBound( );
7. For each tree T in forest F
8. (T.kdom , T.nonk-domfullsky) <--k-domskyfinder(Q, T.getSubtrees,remoteFilter);
9. Probablek-dom= Probablek-dom U T.kdom;
10. Probablek-dom= Probablenonk-domfullsky U T.nonk-domfullsky;
11. K-dom,fullsky oneScanForNonLeaf( )

In step1 of the above mentioned algorithm, we use Compute\_MBR ( ) function to compute the MBR’s of each site and return to the originator. The originator server then forms the tree based on the dominance relation between different MBR. As mentioned earlier, some MBR may get fully pruned in this process. Then we forward the tree to each node so that each node know to what they are connected. After that we start from the leaf nodes and compute the k-dominant skylines and the full skylines in their local proximity. Now, this result will be forwarded to the parent of the nodes, the parent node will compute its k-dominant skylines and the full skylines along with merging the results from its children. This process will continue till we reach the root or the originator of the query.

**4.2 Tree\_Formation(m[i],sitedetails ) -- Algo:**

1. For all sites Si 
   1. For all sites Sj
   2. If Sj completely dominates Si then remove Si from the list.
   3. Else
   4. Find Sj which maximizes the dominance area of Sj under Si.
   5. Assign that Sj to be parent of Si.

We here are forming the tree based on the dominance area. If any site Si is completely dominated by any other site Sj then the site Si will be completely removed from our query computation else we find a site Sk which maximizes the partially dominance area under the site Si and we will make Si as parent of Sk. because if we have the k-dominant and full skylines of Sk then that can prune some of the points in the Si . continuing this way we will get our tree.

Example in fig1, We can see that m3 is completely dominated by m1. So m3 will be taken out of the computations. M1 and m2 are incomparable and hence can execute in parallel.

**4.3 oneScanForLeaf(filter) -- Algo:**

/\*check the input dataset against the filter if it is dominated prune them\*/

1. initialize T=ϕ and R=ϕ and inputdata=localdataset;
2. run conventional onescan algo for k-dom;
3. return k-dom, full-sky.

Here we are just executing the conventional one scan algorithm for finding the k-dominant skylines and full skylines in the leaf nodes or sites in the tree.

**4.4 oneScanForNonLeaf ( inputdata**, **T**) **Algo:**

/\* here we are initializing t and inputdata to be the partial k-dom and full-skyline of the child nodes so as to merge them into the result \*/

1. run conventional onescan algo for k-dom;
2. return k-dom, full-sky.

In this case also we are applying the conventional one scan algorithm but with a slight modification. We here are initializing the T set to contain the partial k-dominant skylines from the child sites and R to contain the partial sky from the child sites so as to avoid the separate merging of the results from the child sites into the parent site.

Evaluation and result:

\begin{algorithm}[t]

\caption{k-dom Sky-Algo}

\label{alg:indexing}

\begin{algorithmic}[1]

\Require Database $D$, Query $Q$

\Ensure K-Dominant dataset $k-dom$

\State for all sites $m[i]\gets$Compute$\_$MBR ( );

\State $T[]\gets$ Tree$\_$Formation($m[i]$,$sitedetails$ );

\State Push tree to each site

\State k-dom= k-domskyfinder( )

\State \Return $k-dominant dataset$

\end{algorithmic}

\end{algorithm}

And refer as Algorithm \ref{alg:indexing}.

The algorithm $1$ will run at the originator of the query.

In step1 of the above mentioned algorithm, we use Compute\_MBR ( ) function to compute the MBR’s of each site and return to the originator. The originator server then forms the tree based on the dominance relation between different MBR. As mentioned earlier, some MBR may get fully pruned in this process. Then we forward the tree to each node so that each node know to what they are connected. After that we start from the leaf nodes and compute the k-dominant skylines and the full skylines in their local proximity. Now, this result will be forwarded to the parent of the nodes, the parent node will compute its k-dominant skylines and the full skylines along with merging the results from its children. This process will continue till we reach the root or the originator of the query.

\begin{algorithm}[t]

\caption{k-domskyfinder }

\label{alg:indexing}

\begin{algorithmic}[2]

\Require Query $Q$, Forest $F={ T1,T2…..}$,filter

\Ensure$k-dom$,$nonk-domfullsky$

\State If $F$ is empty

\State(local$\_$kdom,local$\_$nonk$-$domfullsky)localoneScan(filter)

\State Else Localnonk-domfullsky= localoneScan (filter);

\State remoteFilter =localMBR.getUpperBound( );

\State For each tree T in forest F

\State (T.kdom , T.nonk-domfullsky) =k-domskyfinder(Q, T.getSubtrees,remoteFilter);

\State Probablek-dom= Probablek-dom U T.kdom;

\State Probablek-dom= Probablenonk-domfullsky U T.nonk-domfullsky;

\State K-dom,fullsky= oneScanForNonLeaf( )

\State \Return $k-dom,fullsky$

\end{algorithmic}

\end{algorithm}

And refer as Algorithm \ref{alg:indexing}.

\begin{algorithm}[t]

\caption{Tree$\_$Formation}

\label{alg:indexing}

\begin{algorithmic}[3]

\Require MBR $m[i]$, $sitedetails$

\Ensure forest $F$

\State For all sites Si

\State For all sites Sj

\State If Sj completely dominates Si then remove Si from the list.

\State Else

\State Find Sj which maximizes the dominance area of Sj under Si

\State Assign that Sj to be parent of Si.

\State $A \gets$ Search($Q$, $D$)

\State \Return $A$

\end{algorithmic}

\end{algorithm}

And refer as Algorithm \ref{alg:indexing}.