CS615: Group 7
K-Dominant Skylines in Distributed System

Manish Panwar
Student Id 17
Roll Number 16111040
manishp@iitk.ac.in
Dept. of CSE

Utsab Banerjee
Student Id 21
Roll Number 16111055
utsabb@iitk.ac.in
Dept. of CSE

Indian Institute of Technology, Kanpur

Final report
21st November, 2016

Abstract

Skyline queries are very useful when it comes to filtering out sub-optimal solutions. However, with increase in number of dimensions,the number of skyline points increases as it becomes very difficult to dominate a point in all dimensions. To counter this problem k-dominant skylines have been introduced, where the condition for dominance changes a bit. For a point to be a k-dominant skyline it should not be dominated by another object in any k dimensions. The existing algorithms work well when all the data is present in a single location. In this project we compute k-dominant skyline when the data is distributed over multiple systems. Our experiments show that the algorithm performs very well when the number of k-dominant skylines is low. Additionally, the central server where the computation is done, needs to spare only a small amount of space for the entire computation.

1 Introduction and Problem Statement

Given a set of nodes $N = \{N_1, N_2, ...N_r\}$, where each of nodes contains some objects $DB = \{DB_1, DB_2, ...DB_r\}$, our goal is to find the overall k-dominant skyline set ,such that the communication cost is minimized. The notation SKY_k is used to denote the k-dominant skylines
If we compute the k-dominant skylines in a naive way,we would have to send all the objects from each of the nodes to the server in order to compute the k-dominant skyline. Not only is this an expensive process in terms of cost but the storage also becomes an issue. We propose an algorithm that at any point of time stores at most r elements in the server. The preference function is always assumed to be lesser than.

Assume a scenario where the entire data-set is so large that it cannot be made to fit into a single storage system. Therefore the naive way would not work as it assumes that the entire data is available in a single server. Our algorithm gets rid of this handicap by ensuring that the server stores only r elements,where r is the number of nodes present. This helps us solve the problem practically and the dependence on the size of the entire data-set can be negated.
Even in cases where the data-set is huge but the k-dominant skylines are less,we save an huge amount of communication thereby making the algorithm faster and suitable for practical purposes.

1.1 Related Material

Previous studies on skyline query,k-dominant skyline and computing skylines in distributed system are discussed in brief.

1.1.1 Skyline Query Processing

Each object in database D has d' attributes. Skyline query is over d $\leq d'$ attributes, called *skyline attributes*. Informally, return all objects in database that are *not dominated* by any other object. For each skyine attribute,

a dominating function is defined

- Can be any comparison operator, denoted by $\succ$
- If it is at least equal,then it is denoted by $\succeq$

An object O_p **dominates** another object O_q if $O_p \succ O_q \iff \forall i, O_{pi} \succeq O_{qi}$, and $\exists j, O_{pi} \succ O_{qi}$.

The **skyline query** returns the set of objects S $\subseteq$ D such that for all objects $O_p \in$ S and all objects $O_p \notin$ S, $O_p \in S \iff \nexists O_q \in D, O_q \succ O_p$

1.1.2 K-Dominant Skyline

In high dimensions, it becomes extremely difficult for an object to dominate another in all ddimensions. number of skyline becomes too impractical.
K-dominance: It is enough for an object to be preferred in k¡d dimensions to dominate another object.
K-dominant skylines are those that are not k-dominated by any other object in any subset of size k¡d. Ther are different from skylines of dimensionality k,and are basically the intersection of such k-dominant skylines.

1.1.3 Computing Skylines in Distributed systems

With the development of big data and cloud computing, data storage has trended to become increasingly distributed. In practice, more and more applications need to collect data from multiple data sources which have distributed and decentralized control. Since the data uncertainty and distributed data storage have been the inherent characteristics of many applications, DSUD algorithm was designed to work well for such systems. The general framework is as follows

1. **To-server**
   - Each P_i sends tuple t_i in S_i with the *largest* skyline probability to C
2. **Server-calculation**
   - C updates skyline probabilities of all tuples t_i by constituting them as one dataset
3. **Server-delivery**
   - C selects t_0 with largest *partial* skyline probability
   - C broadcasts t_0 to all sites(except its origin, say P_i)
   - Global skyline probability of t_0 is, thus, computed
   - In the beginning of next round, P_i will send its next largest tuple to C
4. **Local-pruning**
   - t_0 is used to prune skyline sets S_j

2 Algorithm or Approach

We use a filter bases algorithm to compute the k-dominant skylines efficiently. The algorithm mainly comprises of two parts, sorting each of the datasets based on some scoring function and finding the overall k-dominant skylines.
We use two scoring functions which are described as follows

A. **Sort By Domination Power**
To prune unnecessary objects efficiently in the k-dominant skyline computation, we compute domination power of each object. An object is said to have δ-domination power if there are δ minimal values in which it is better or equal to all other objects of DB. We sort objects in descending order by their values of domination power (δ). If more than one objects have same domination power then sort those objects in ascending order of the sum value. This order reflects how likely an object k-dominate other objects. Higher objects in the sorted sequence are likely to dominate other objects. Thus this pre-processing helps to reduce the computational cost of k-dominant skyline. Experiments show that our estimation is robust over various distributions. Moreover, it also works well when data values are correlated, independent or anti-correlated.

B. **Sort By Best-k Sum**
By using the ordered objects, we can eliminate some of non-skyline objects easily. To get the benefit of ordered objects many skyline algorithms have used a sum of the values of the dimensions for each objects. However, this sort is not effective for k-dominant query computation especially when values of each attribute is not normalized. For example, assume O_i=(1, 2, 3, 3, 3, 2) and O_j=(7, 1, 3, 2, 3, 1) are two objects in 6-dimensional space. Although sum of O_i's values is smaller than that of O_j's, O_i does not 5-dominant of O_j. Instead, O_i is 5-dominated by O_j.

So instead of taking the sum of all the dimensions we take the sum of the best-k dimensions. In the example above if we take the sum of the best k(best 5 as in example) then O_j would have appeared before O_i in the sorted sequence. Thereby the best-k sum gives us a better insight about the objects than just simply taking the sum over all the dimensions.

Obj.	D1	D2	D3	D4	D5	D6	DP	Sum
O_7	1	2	5	3	1	2	4	14
O_5	1	4	1	1	2	4	3	13
O_4	5	3	5	4	1	2	2	20
O_2	3	4	4	5	1	3	1	20
O_6	5	3	4	5	1	5	1	23
O_3	4	3	2	3	5	4	0	21
O_1	7	3	5	4	4	3	0	26

Table 1: Objects sorted according to scoring function A

Obj.	D1	D2	D3	D4	D5	D6	KSum
O_5	1	4	1	1	2	4	9
O_7	1	2	5	3	1	2	9
O_2	3	4	4	5	1	3	15
O_4	5	3	5	4	1	2	15
O_3	4	3	2	3	5	4	16
O_6	5	3	4	5	1	5	18
O_1	7	3	5	4	4	3	19

Table 2: Objects sorted according to scoring function B

As we can see that while computing the 5-dominant skylines we get the final set as $SKY_5 = \{O_5, O_7\}$. These two tuples are the ones that appear on top when the two scoring functions are used.

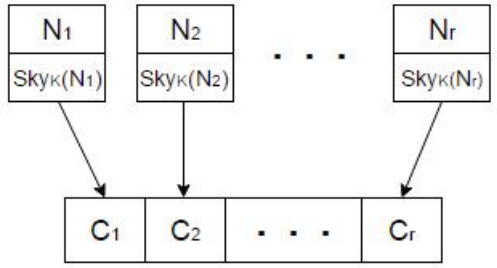

Figure 1: Graphical representation of the Distributed setup

Algorithm 1 k-Dominant Skyline in Distributed System

initialize $Sky_k(server)$=ϕ
for *each node* N_i **do**
  Compute k-dominant skyline $Sky_k(N_i)$ in each of the distributed systems and sort them by any of the above scoring function
end
for *each node* N_i **do**
  N_i sends to the server tuple t_i from $Sky_k(N_i)$ which has best scoring power
end
while *for all i,* $Sky_k(N_i) \neq \phi$ **do**
  Server computes the best among these new tuples received by constituting them as one dataset
  The server then chooses the tuple t_s with best scoring power and broadcasts it to all the nodes, except the one it had received from (say N_j)
  This tuple can dominate some k-dominant skyline in other nodes, hence reducing the computation set
  Some non k-dominant skyline can also dominate this tuple.
  if *tuple* t_s *is not k-dominated by any tuple* **then**
    Add t_s to $Sky_k(server)$
  end
  if $Sky_k(N_j) \neq \phi$ **then**
    N_j sends the next best tuple to server.
  end
end
for *each tuple* t_i *in the server* **do**
  The server broadcasts it to all the nodes, except the one it had received from
  if *tuple* t_i *is not k-dominated by any tuple* **then**
    Add t_i to $Sky_k(server)$
  end
end

Finally the set $Sky_k(server)$ contains all the k-dominant skylines.

3 Results

The experiments done on various data-sets give us the following results. First we look into an independent data-set and the variations in the number of messages passed when the value of k changes. The naive method has a constant number of messages sent which is equal to the total number of elements in the entire data-set.

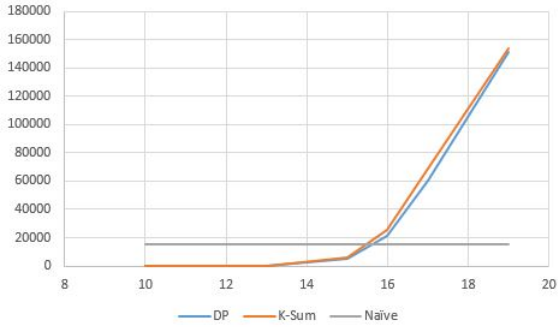

Figure 2: Graphical representation of Q

Now we run our algorithm in this data-set of correlated data. The total number of nodes were 10 and the sum of all the objects in the data-sets in each node was 10000. The computation was done on 20 dimensions in varying values of k.

Values of k	Correlated Data				
		Scoring function A		Scoring function B	
	Skyline	Message	Time	Message	Time
10	1	25	0.00074	25	0.00074
12	2	38	0.00149	66	0.00158
14	3	60	0.00293	179	0.00276
16	17	230	0.01611	757	0.01542
18	150	1768	0.13450	3266	0.13560
19	490	5521	0.4548	8089	0.4681

Figure 3: Graphical representation of Q

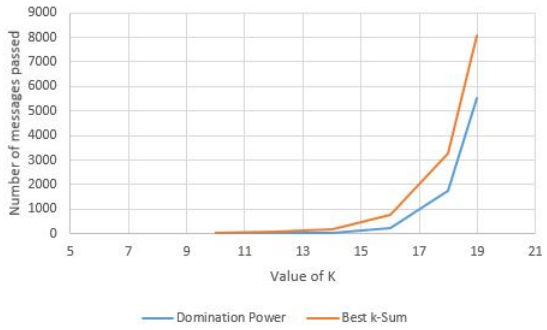

Figure 4: Graphical representation of Q

Now we run our algorithm in this data-set of anti-correlated data. The total number of nodes were 10 and the sum of all the objects in the data-sets in each node was 10000. The computation was done on 20 dimensions in varying values of k.

Values of k	Anti-Correlated Data				
		Scoring function A		Scoring function B	
	Skyline	Message	Time	Message	Time
10	0	10	0.00004	10	0.00003
12	0	14	0.00022	14	0.00019
14	88	2078	0.138	2389	0.141
16	3214	35905	3.723	40272	4.167
18	8455	93083	12.507	96780	12.7117
19	9985	109835	15.302	109863	14.963

Figure 5: Graphical representation of Q

Figure 6: Graphical representation of Q

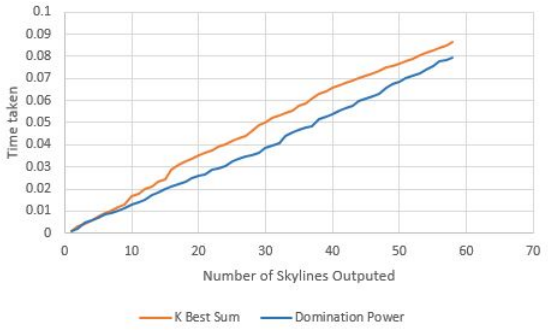

Figure 7: Graphical representation of Q

In this figue we see how the progressiveness is maintained by this algorithm. The number of skylines generated are on the x axis and the time in on y. If we use the naive algorithm then all the k-dominant skylines can be reported only after the entire computation is done and therefore violates progressiveness.

Analysing the algorithm, a skyline can only be reported after it has been sent to all the r nodes. So the number of messages sent is always more that r times the of k-dominant skylines. Any algorithm that works in such a setup can do no better as these messages cannot be minimized.
The time taken by the algorithm will also decrease as we

have simply simulated the distributed setup but have not had the power to process parallely. Therefore the actual time would go down since the comparison could have been done very quickly.

4 Conclusions

As we have clearly seen in the results section that the algorithm performs extremely well on correlated data or when the number of k-dominant skylines in less. The main advantage of the algorithm lies in the fact that it is progressive and can be used to find k-dominant skylines in data where the entire data cannot fit into any memory. Progressiveness helps in quickly generating some results and computing the rest while the user can work on with the already generated data.

There are a number of things we can work on in future and the most important being trying out new scoring functions to sort the data. The variations in the scoring functions mentioned makes it an interesting area to explore. Trying to find out scoring functions that will work well with various kinds of data can make this algorithm perform a lot better.

References

[1] S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator," in ICDE, 2001, pp. 421-430.

[2] C. Y. Chan, H.V. Jagadish, K-L. Tan, A-K. H. Tung, and Z. Zhand, "Finding k-Dominant Skyline in High Dimensional Space," in Proceedings of ACM SIGMOD, 2006, pp. 503-514.

[3] X. Ding and H. Jin, "Efficient and progressive algorithms for distributed skyline queries over uncertain data" IEEE Trans. Knowl. Data Eng., vol. 24, no. 8, pp. 1448-1462, Aug., 2012