Scalable Processing Of Dominance-Based Queries

Georgitsis Christos
cgeorgit@csd.auth.gr

Dialektakis Georgios
gdialekt@csd.auth.gr

Author Keywords
Big Data; Skyline; Apache Spark; Top-k; Dominating Queries; Distributed Environments

INTRODUCTION

Living in the Information Age allows almost everyone has access to a large amount of information and options to choose from in order to fulfill their needs. The rapid growth of decision support systems and the increasing size of multidimensional data lead researchers to search for new efficient methods for data processing in order to retrieve useful insights. Various advanced analytical methods such as mathematical models, statistical analysis, and data mining have been used for retrieving useful information. Moreover, during the last decades, data management and storage have become increasingly distributed. Advanced query operators are necessary in order to help users to handle the huge amount of available data by identifying a set of interesting data objects. In this paper, we study the skyline query, a decision support mechanism, and Top-k queries, a rank-aware approach. Skyline query retrieves the value-for-money options of a dataset by identifying the objects that present the optimal combination of the dataset's characteristics [5]. On the other hand, Top-K queries define a cumulative scoring function in order to retrieve the best results of a dataset since this will reduce the potential multidimensional comparisons of data to a single scalar value.

Skyline

Skyline query processing in highly distributed environments poses inherent challenges and demands and requires non-traditional techniques due to the distribution of content and the lack of global knowledge [4]. The popularity of the skyline operator is mainly due to its applicability for multi-criteria decision-making applications [6]. For simplicity, we assume that skylines are computed concerning minimum conditions on all dimensions; however, all methods discussed can be applied with any combination of conditions. Skyline operation is a well-established technique that filters out a set of interesting points from a potentially large set of data points. A point is interesting if it is not dominated by any other point [7]. Let D be a d-dimensional database. Given a set of points $\{p_1, p_2, ..., p_n\}$, a point p_i is said to dominate another point p_j if p_i is not worse than p_j in any of the d-dimensions and p_i is better than p_j in at least one of the d-dimensions. Then, the skyline query returns some points, each of which is not dominated by other points in D. The skyline of a d-dimensional dataset contains the points that are not dominated by any other point on all dimensions. Skyline computation has recently received considerable attention in the database community, especially for progressive methods that can quickly return the initial results without reading the entire database [6].

ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

For example, consider a database that contains information about laptops. Each tuple of the database is represented as a point in a data space consisting of numerous dimensions (such as the laptop's price, the laptop's weight). Assume a user is looking for laptops with low price and as close as possible to lightweight. In this case, it is not obvious if the user would prefer: a laptop with a cheap price but more heavyweight or a very lightweight but farther more expensive laptop. The skyline query retrieves all laptops for which no other laptop exists that is cheaper and less weighted. A skyline query does not require a scoring function defining the relative importance of all criteria [4].

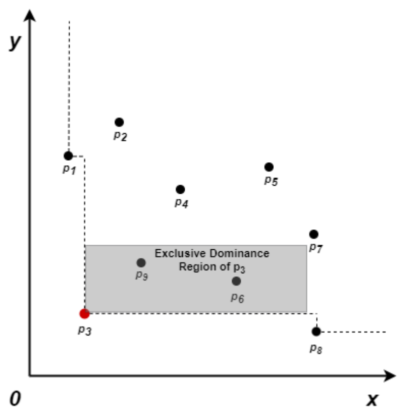
Top-k Dominating Query

Top-K queries retrieve the best k objects that minimize a specific preference function. The difference from skyline queries is that the output changes according to the input function, and the retrieved points are not guaranteed to be part of the skyline [6]. Top-k dominating queries combine the advantages of top-k queries (the number of results is bounded) and skyline queries (no parameters and user-defined scoring functions are required), eliminating their disadvantages by assigning to each object an intuitive score based on dominance. This score is reflecting the importance of every object in a dataset in a natural way [8]. For this reason, top-k is important in supporting multi-criteria decisions [1].

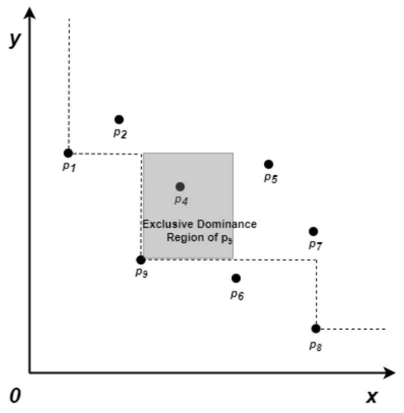
For example, consider the same database that contains information about laptops. Each tuple of the database is represented as a point in a data space comprising numerous dimensions (such as the laptop's price, the laptop's weight). Assume a user is looking for laptops with low price and as close as possible to lightweight. Here, it is not obvious if the user would prefer: a laptop with a cheap price but more heavyweight or a very lightweight but farther more expensive laptop. The top-k dominance query searches among all laptops and yields as a result k laptops that provide the best combination of low price and lightweight in ascending order, i.e., top-1 being the best and top-k being the worst among those k laptops.

IMPLEMENTATION

In this chapter, we present the methods that we used to solve the three tasks of this work, the Skyline set, the Top-k dominating points and the Top-k dominating Skyline points, respectively.



(a)



(b)

Figure 1: Example of top-k dominating query using STD algorithm.

Task 1

The first task of this work is, given a set of d-dimensional points, we need to find the set of points that are not dominated. This is also known as the skyline set. To compute the skyline set, we implemented a distributed and improved version of the Block-Nested-Loop Algorithm (BNL) [2], named Sort-Filter-Skyline (SFS) Algorithm [3]. The first step of SFS comes after its name and requires sorting the dataset. Presorting enforces that a point p dominating another point q will be visited before q. In our work, we sort the input dataset in descending order, according to the sum of the points' coordinates on all dimensions. Next, we perform a local skyline query, which calculates the skyline set of each partition of the dataset that the Driver has distributed to each worker node. The SFS algorithm keeps a window in the main memory, which holds the skyline points that have been found so far. The SFS algorithm consists of the following steps:

- Insert the first point of the dataset into the empty window.
- For each point in the window read the next point p from the dataset.
- If point p dominates a point q from the window, remove q.
- If point p is dominated by a point from the window, stop comparing p to other points in the window and discard it.
- If point p is not dominated by any point from the window, insert p into the window.
- Repeat from step 2 for all points in the dataset.
- Return the window which contains the skyline points.

The execution of the local skyline query terminates when all worker nodes return the computed skyline set to the Driver. Finally, after collecting the results from each worker, the Driver finds the final skyline set by executing the above steps of the SFS algorithm. However, unlike the local skyline query, the algorithm's input is the points from the local skylines computed by the workers. An example can be shown in figure 1a, where the skyline set consists of the points $\{p_1, p_3, p_8\}$ as they are not dominated by any point of the dataset.

Task 2

The second task of this work is to find the k points with the highest dominance score of a dataset composed of d-dimensional points. This is also known as the top-k dominating query. For this task, we used a method that depends on the skyline query, named Skyline-Based Top-k Dominating Algorithm (STD) [6]. However, instead of using the Branch-and-Bound Skyline Algorithm (BBS), as proposed in [6], we applied the SFS algorithm as discussed in the previous section. The steps followed by STD are:

- Find the skyline set S of the dataset D.
- For each point $p \in S$ count the number of points it dominates (i.e., find its dominance score)
- Sort the points in S by their dominance scores. Report the point p with the highest score (top-1).
- Next, the exclusive dominance region R of p is computed and a local skyline query in R is executed. The exclusive dominance region of a point p of S consists of the points dominated only by q and not by any other point of S. After the local skyline query is executed, p is removed from the list that contains the skyline points with their dominance scores.
- Calculate the dominance scores of all skyline points in R and updated the sorted list.
- Repeat the above precess from step 3 k times until all top-k dominating points are detected.

It is important to mention that the dominance score of the skyline points is computed by broadcasting the skyline set to all worker nodes. Then, each worker maps every point p to a $(p, 1)$ pair if p dominates another point, or $(p, 0)$ if it does not. Finally, the total dominance score is calculated by aggregating the dominance score of the above pairs using their sum as the reduce function.

We illustrate the process of finding the top-3 dominating points for the dataset shown in figure 1, using the above steps of the

Dimension	2	3	4	5
Dataset Size	10K	100K	1M	10M
Executors	1	2	4	8
k	-	10	100	-

Table 1: Parameters

STD algorithm. The skyline S of D is $S = \{p_1, p_3, p_8\}$. The domination scores of the points in S are computed and stored in a list which is then sorted: $\{< p_3, 6 >, < p_1, 1 >, < p_8, 0 >\}$. Then, the exclusive dominance region R of p_3 is obtained (figure 1a), p_3 is reported as top-1, and then is removed from the list. A constrained skyline query is performed in R, which contains $\{p_6, p_9\}$, their dominance score is found which is $dom(p_6) = 2$ and $dom(p_9) = 3$, and the sorted list is updated: $\{< p_9, 3 >, < p_6, 2 >, < p_1, 1 >, < p_8, 0 >\}$. A local skyline query is then performed in the exclusive dominance region of p_9 (figure 1b), p_9 is reported at top-2 and is removed from the list. The point p_4 with $dom(p_4) = 1$ is inserted in the list which becomes: $\{< p_6, 2 >, < p_1, 1 >, < p_4, 1 >, < p_8, 0 >\}$. The point p_6 has the highest dominance score in the list, so it is reported as top-3, and the STD algorithm terminates.

Task 3

The final task of our work is to find the top-k dominating points that belong to the skyline set. For this reason, we first calculate the skyline of the whole dataset and store the result into an array. We then compute the dominance score for each skyline point in the same way as in task 2 and store it in an array which consists of the points and their score. Next, we sort the array in descending order by their dominance score. Finally, we select the first k elements of this array which make up the top-k dominating points of the skyline.

EXPERIMENTS

Setup

We conducted several experiments, using Apache Spark, to evaluate the performance and scalability of our algorithms in the three tasks, as discussed in the previous section. Our experiments were performed locally on two different machines, with the first one having four physical cores, and the other one having eight physical cores. The input of our algorithms was a set of d-dimensional points, where the coordinates of each point were double numbers. Those doubles were generated in random, based on four different distributions (correlated, uniform, normal, anticorrelated), as shown in figure 2. During our tests, we examined the performance of the implemented algorithms and how it changes with respect to various parameters. In specific, the parameters we explored are the size of the input data, the number of dimensions which depends on the input dataset, the parallelism, i.e., the number of Spark worker nodes (executors), the distribution of the input data, and the user-defined parameter k for the second and third task. Table 1 shows the different values that each parameter can take.

Task 1 Results

In figure 3a, the execution time of the skyline query is roughly the same between the four distributions. Also with the increase of the points' dimensionality there is a small increase in the execution time as the number of coordinates increase and the algorithm has to perform more comparisons. Moreover, we observe that the execution time of this algorithm remains almost the same with respect to the increase of the dataset size. However, when we enlarge the dataset size to 10 million points, the execution time rises rapidly, as shown in figure 3b. As we expect, increasing the parallelism of the system by adding more executors, the execution time of the skyline algorithm drops, showing scalability (figure 3c).

	k	**Correlated**	**Uniform**	**Normal**	**Anticorrelated**
Task 2	10	143.5	165.5	168.2	139.0
	100	1135.1	1300.2	1245.8	1065.3
Task 3	10	61.5	101.5	99.09	56.6
	100	62.4	103.7	104.9	58.1

Table 2: Execution time of task 2 and task 3 with respect to parameter k.

Task 2 Results

Figure 4 illustrates the performance of the top-k dominating query. We observe that the execution time of the STD algorithm remains quite low for 2-d and 3-d points. On the other hand, the execution time for 4-d points is twice as much as the 2-d and 3-d points. Furthermore, when each point of the dataset has 5 dimensions, the query takes up to roughly 3 times the time of 3-d data points, as depicted in figure 4a. As we can see in figure 4b, the top-k dominating query takes about the same time to complete regardless the size of the dataset. However, when the algorithm receives 10 million points as input, the execution time is more than 7 times larger than the time in the previous three cases. In figure 4c, we notice the scalability of the STD algorithm, since the time decreases as the task is distributed to more executors. Finally, in table 2 the performance of the top-k dominating query is summarized, where we observe an enormous increase in execution time for $k = 100$ compared to $k = 10$. This is mainly due to the complexity of the STD algorithm which repeatedly performs skyline queries based on the value of the parameter k.

Task 3 Results

Figure 5 illustrates the performance of the skyline top-k dominating query. We observe that the execution time remains quite the same for 2-d and 3-d points, without notable differences for each dimension. On the other hand, the execution time for 4-d points increases remarkably in contrast with the 2-d and 3-d points. Similar behavior is observed in the 5 dimensions, since in this case the time increases significantly, as depicted in figure 5a. As we can see in figure 5b, the top-k dominating query takes about the same time to complete regardless the size of the dataset. However, when the algorithm receives 10 million points as input, the execution time is more than 8 times larger than the time in the previous three cases. In figure 5c, we notice there is a decrease in the execution time as the task is distributed to more executors, something that confirms the distribution of our work. In table 2, in contrast to task 2, the execution time of task 3 remains constant regardless of the value of parameter k, as the algorithm calculates the

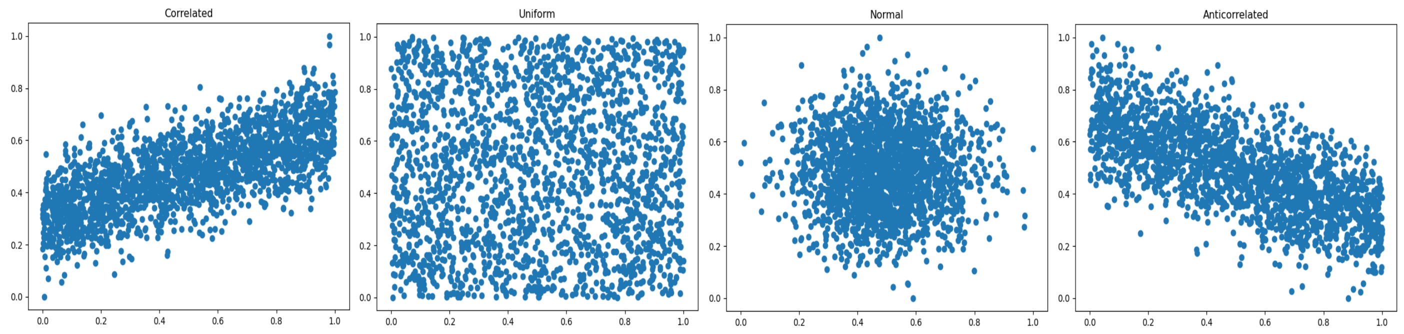
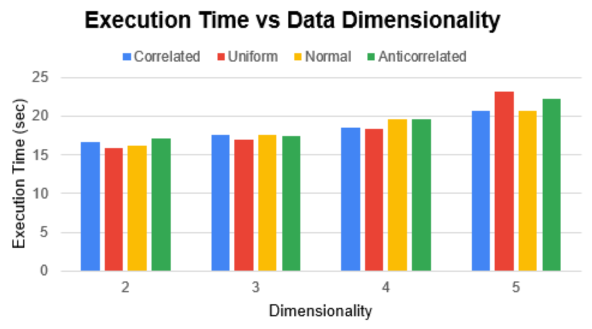
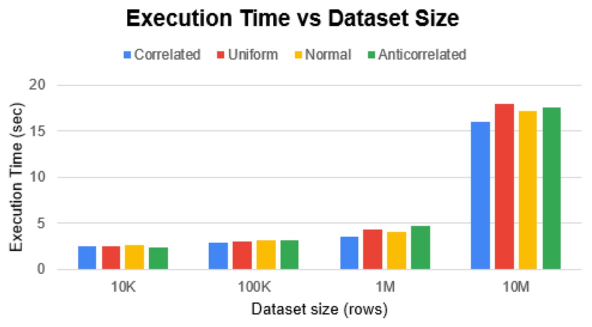


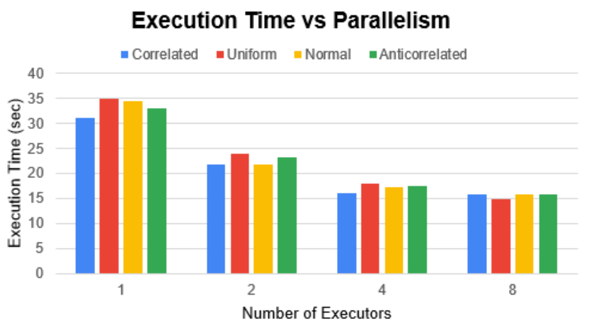
Figure 2: Different distributions for 2-d data.



(a) Experiments performed with 10M dataset and 4 executors.

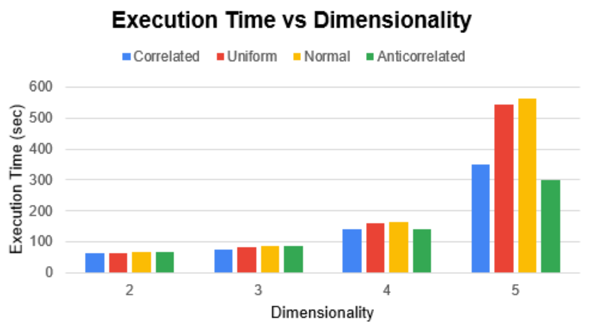


(b) Experiments performed with 4-d data and 4 executors.

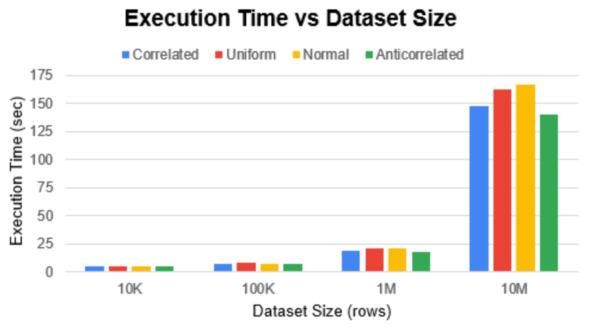


(c) Experiments performed with 4-d data and 10M dataset.

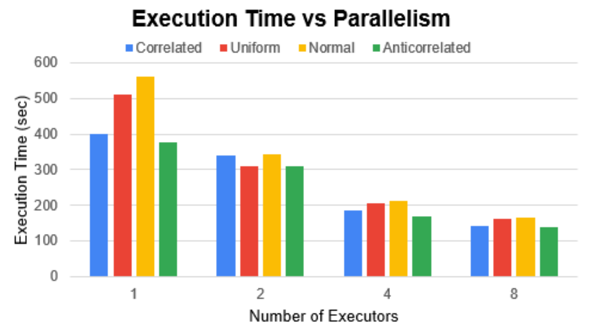
Figure 3: Execution time results of task 1.



(a) Experiments performed with 10M dataset and 8 executors.



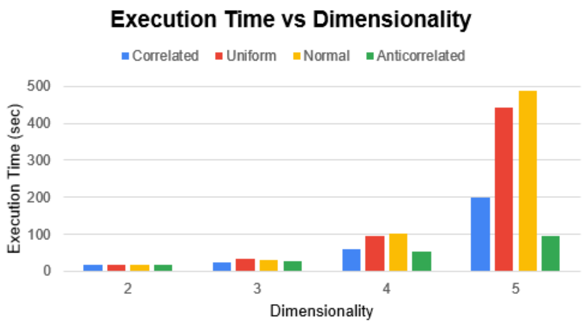
(b) Experiments performed with 4-d data and 8 executors.



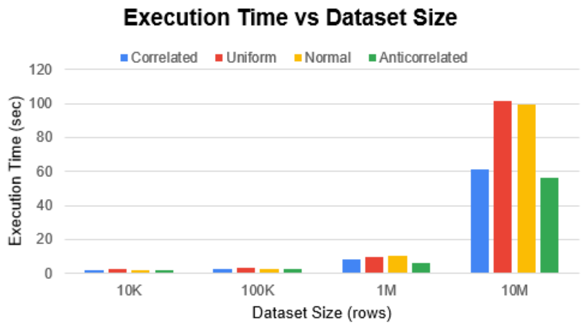
(c) Experiments performed with 4-d data and 10M dataset.

Figure 4: Execution time results of task 2

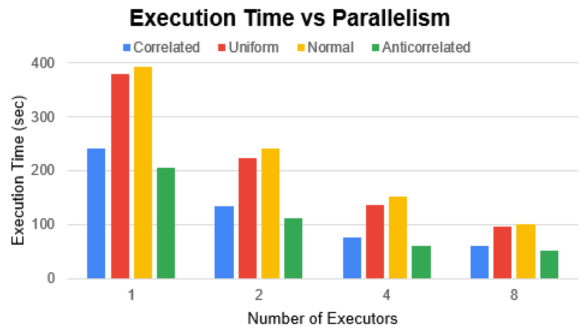
dominated points for each point that belongs to the skyline and there is no need to search the whole dataset for top-k points.



(a) Experiments performed with 10M dataset and 8 executors.



(b) Experiments performed with 4-d data and 8 executors.



(c) Experiments performed with 4-d data and 10M dataset.

Figure 5: Execution time results of task 3

CONCLUSION

During the last decades, the vast number of independent data sources and the high rate of data generation have made a centralized assembly of data at one location infeasible. As a consequence, data are increasingly stored in a distributed way, therefore distributed query processing has become an important and challenging problem. In this study, we investigated the important and challenging problems of skyline query and top-k dominating queries in distributed environments. The popularity of the skyline operator is mainly due to its ability to identify a set of interesting objects in a large database. Moreover, the Skyline operation is useful for a number of database applications, including decision support and visualization. On the other hand, Top-k dominating queries combine the advantages of regular top-k and skyline queries, by bounding the size of the result without the need for user-defined scoring functions. In the last years, they became an active research area due to their importance in several modern applications. Finally, for the evaluation of each task, we ran various experiments on synthetic data, in order to first ensure that our algorithms run in parallel and secondly to come in contact with the possibilities offered by parallel programming. The results show that our submitted methods significantly reduce the communication cost and latency compared with the baselines.

REFERENCES

[1] Daichi Amagata, Yuya Sasaki, Takahiro Hara, and Shojiro Nishio. 2016. Efficient processing of top-k dominating queries in distributed environments. *World Wide Web* 19, 4 (2016), 545–577.

[2] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proceedings 17th international conference on data engineering*. IEEE, 421–430.

[3] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with presorting. In *ICDE*, Vol. 3. 717–719.

[4] Katja Hose and Akrivi Vlachou. 2012. A survey of skyline processing in highly distributed environments. *The VLDB Journal* 21, 3 (2012), 359–384.

[5] Christos Kalyvas and Theodoros Tzouramanis. 2017. A survey of skyline query processing. *arXiv preprint arXiv:1704.01788* (2017).

[6] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)* 30, 1 (2005), 41–82.

[7] Borzsonyi Stephan, Kossmann Donald, and Stocker Konrad. 2001. The skyline operator. *Proc. of (ICDE01)* (2001), 421–430.

[8] Eleftherios Tiakas, A. Papadopoulos, and Y. Manolopoulos. 2015. Top-k Dominating Queries : a Survey.