Scalable Processing of Dominance-Based Queries

Skyline query and *top-k* dominating queries

|  |  |
| --- | --- |
| Emmanouil Kalyvas  School of Informatics  Aristotle University  Thessaloniki, Greece  kalyvase@csd.auth.gr | Vasileios Ntaoulas  School of Informatics  Aristotle University  Thessaloniki, Greece  ntaoulasv@csd.auth.gr |

ABSTRACT

The information age has brought about unprecedented amounts of data, which has in turn led to the development of big data and the use of analytics to make sense of it. Data management has become a critical aspect of this process, as it involves organizing and storing data in a way that makes it accessible and usable. Query operators are a type of tool that allow users to retrieve specific data from a database by specifying certain criteria. Overall, the proliferation of data and the ability to analyze it has had a huge impact on a variety of fields and is likely to continue to shape the way we live and work in the future. 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 [1]. 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.

KEYWORDS

Scala; Spark; Big Data; Dominance; top-k; skyline; queries

1 INTRODUCTION

As decision support systems have grown in popularity and multidimensional data has become larger, researchers have sought ways to efficiently process data and gain insights from it. Operational research, which involves using advanced analytical methods such as mathematical models and data mining to support decision making, is one area that has developed a number of rank-aware approaches, including those used in top-k queries. However, these approaches can sometimes be limited by their reliance on a single scoring function, which may not capture the full complexity of multi-dimensional data.

Skyline queries offer an alternative approach that is more intuitive for humans to understand. Unlike *top-k* queries, which use specific ranking functions and criteria, skyline queries consider the preferences of users over the attributes of the data. These preferences, which might include things like personal likes and dislikes or preferred vacation destinations, are used to identify the subset of data that is most interesting and preferred based on the preferences of all users. This subset, known as the skyline set or pareto optimal set, represents the most interesting and preferred items in the dataset.

In recent years, the processing of skyline queries has become a significant topic in database research, as they are useful for extracting interesting objects from multi-dimensional datasets in a way that takes into account the preferences of users. The simplicity and applicability of the skyline operator to multi-criteria decision support based on user preferences have made it a popular choice for many applications.

1.1 Skyline

Skyline processing was first studied in a single-database environment, i.e., in a centralized setup. As nowadays data are increasingly stored and processed in a distributed way, skyline processing over distributed data has attracted much attention recently [2]. They have been an active area of research in computer science, and over the years, there have been numerous advances in the algorithms and techniques used to support skyline queries. These advances have made it possible to process large datasets more efficiently, and to support more sophisticated preference criteria.

Some of the most important benefits of using skyline queries are:

* They allow users to easily identify the most preferred objects in a dataset, based on multiple criteria.
* They can be used to personalize search results, by allowing users to specify their own preference criteria.
* They can be used to support decision making by presenting a ranked list of the most preferred options.
* They do not require the use of a specific ranking function; their results only depend on the intrinsic characteristics of the data.

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 [3]. For example, consider a database that contains information about cars. Each tuple of the database is represented as a point in a data space consisting of numerous dimensions (such as the car’s price, the car’s highspeed etc.). Assume a user is looking for cars with low price and not so old. In this case, it is not obvious if the user would prefer: a car with a low price but older, or a new one but more expensive. The skyline query retrieves all cars for which no other car exists that is cheaper and less old.

A more clear definition of skyline has been given in [2]:

(Skyline): A point p ∈ S is said to dominate another point q ∈ S, denoted as p ≺ q, if (1) on every dimension di ∈ D, pi ≤ qi and (2) on at least one dimension dj ∈ D, pj < q j. The *skyline* is a set of points SKY(S) ⊆ S that are not dominated by any other point. The points in SKY(S) are called skyline points.

Fig. 1 shows an example of how skyline points are represented in 2-d space (green dots). Of course, this can be extended for more dimensions.

Chart, scatter chart

Description automatically generated

**Fig. 1:** Example of 3 skyline points

1.2 *Top-k* dominating query

Top-k dominating queries retrieve the *k* data objects that dominate the highest number of data objects in a dataset. The *top-k* dominating query was first introduced by Papadias [4] as an extension of the skyline query, but nowadays they have become an important tool for decision support, data mining, web search, and multi-criteria retrieval applications and have also been studied by different perspectives, such as in indexed and non-indexed multi-dimensional data using efficient exact computation algorithms, in uncertain data using randomized algorithms and in data streams [5].

As a *top-k* query, the user can bound the number of returned results through the parameter *k* and provide a ranking score function (usually monotone) to order the objects by their scores and, thus, retrieve the top-k best objects. They do not provide an objective order of importance for the points, because their results are sensitive to the preference function used [6]. For example, consider the same database that contains information about cars. Each tuple of the database is represented as a point in a data space comprising numerous dimensions (such as the car’s price, the car’s highspeed etc.). Assume a user is looking for a car with low price and not so old. Here, it is not obvious if the user would prefer: a car with a low price but older, or a new one but more expensive. The *top-k* dominance query searches among all cars and yields as a result *k* cars that provide the best combination of low price and new in ascending order, i.e., *top-1* being the best and *top-k* being the worst among those *k* cars.

The authors of [5] defined *top-k* queries as:

The object p = (p..x1, p..x2, …, p..xd) ∈ D dominates another object q = (q..x1, q..x2, …, p..qd) ∈ D, i.e., p ≺ q, when: ∀i ∈ {1, …, d} : p..x1 ≤ q..x1 Λ ∃i ∈ {1, …, d} : p..x1 < q..x1. This means that p is as good as q in all dimensions, and it is strictly better than q in at least one dimension. Then, the domination score of p, dom(p) is defined as: dom(p) = | {q ∈ S : p ≺ q} |. A top-k dominating query returns the k objects with the maximum domination scores in D.

Fig. 2 shows an example of *top-4* dominant points represented in 2-d space. Of course, this can be extended for more dimensions.

Chart, scatter chart

Description automatically generated

**Fig. 2:** Example of *top-k* dominant points (k = 4)

The rest of the paper is organized as follows: Section 2 reviews the implementation of the case problem, for each given task: Given a potentially large set of d-dimensional points, where each point is represented as a d-dimensional vector, we need to detect interesting points. Section 3 presents the results and, finally, Section 4 summarizes the experiment and concludes the paper.

2 IMPLEMENTATION

In this section we introduce the multi-dimensional problem we were called to solve, using skyline and top-k dominating queries. Given a potentially large set of d-dimensional points, where each point is represented as a d-dimensional vector, we need to detect interesting points. The problem is divided into 3 tasks:

* Task 1: Find the skyline set of the given dataset.
* Task 2: Find the top-k points with the highest dominance score.
* Task 3: Find the top-k skyline points with the highest dominance score.

We produce 4 datasets for 4 different distributions (correlated, uniform, normal, anti-correlated) for *d* dimensions. Fig. 3 shows examples for each distribution for 2-d data.

|  |  |
| --- | --- |
| Chart, scatter chart  Description automatically generated  correlated | Chart, scatter chart  Description automatically generated  normal |
| Chart, scatter chart  Description automatically generated  uniform | Chart, scatter chart  Description automatically generated  anti-correlated |

**Fig. 3:** The 4 distributions for 2-d data

The implementation of each task is described below.

2.1 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 firstly sorted the input dataset in ascending order, according to the sum of the points’ coordinates on all dimensions. We also give an index for each point. We define a skyline function that computes the skyline set given a set of points. This function contains the following logic:

We initialize the first point of the sorted set as skyline point and compare each next point with the current set of skyline points. Specifically:

1. Initialize the first point of the sorted dataset as skyline point.
2. We read the next point *p* from the dataset.
3. If point *p* dominates one of the current skyline points, remove this skyline point from the set.
4. If point *p* is not dominated by any point from the skyline set, insert in into the skyline set.
5. Repeat step-2 for the rest points in the dataset.
6. Return the skyline set.

In order to execute this algorithm in a distributed manner, the dataset is read into an RDD and the skyline function is executed on each of the partitions, to gather the local skyline points of each one of them. We then collect all the skyline points of each partition on the driver and pass them through the skyline algorithm one more time. This gives us the final skyline set of the whole dataset.

2.2 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 apply the following algorithm:

1. Load the dataset *points.*
2. Compute the skyline set with the algorithm described on Task 1 given the dataset *points.*
3. For each one of the skyline points, calculate the number of points they dominate.
4. Create a list *candidate\_points* of candidate top points that is initialized to be equal to the skyline set gathered at step 2 along with their number of dominations.
5. Create an empty list *top\_points* of the top k points.
6. Sort the *candidate\_points* list by the number of dominations in descending order.
7. Get the first element *top\_point* of the *candidate\_points* list, add it to the *top\_points* list and remove it from the dataset *points.*
8. Create a list *dominated\_points* that contains all the points dominated by the *top\_point* excluding itself and the points in the list *top\_points*.
9. Compute the skyline points of the *dominated\_points* list, using the algorithm described on task 1 and count the number of dominations for each one of them.
10. Add the skylines points of the *dominated\_points* list along with their dominations in the *candidate\_points* list.
11. If the length of the *candidate\_points* list is zero or we got the top k points then return *top\_points* list else go to step 6.

2.3 Task 3

The final task of our work is to find the top-k dominating points that belong to the skyline set. The steps are:

1. We keep only the skyline points we computed before.
2. We calculate the dominations of each skyline point.
3. Sort them by descending order.
4. Take the first *top-k* points with the most dominations.

3 RESULTS

We conducted several experiments, using Apache Spark, to evaluate the performance and scalability of our algorithms in the three tasks we introduced before. Our experiments were performed locally using 4 and 8 cores respectively. 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 we saw in Fig. 3. 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 distribution of the input data, and the user-defined parameter k for the second and third task. Table 1 shows the different values that we used for each parameter for the following experiments.

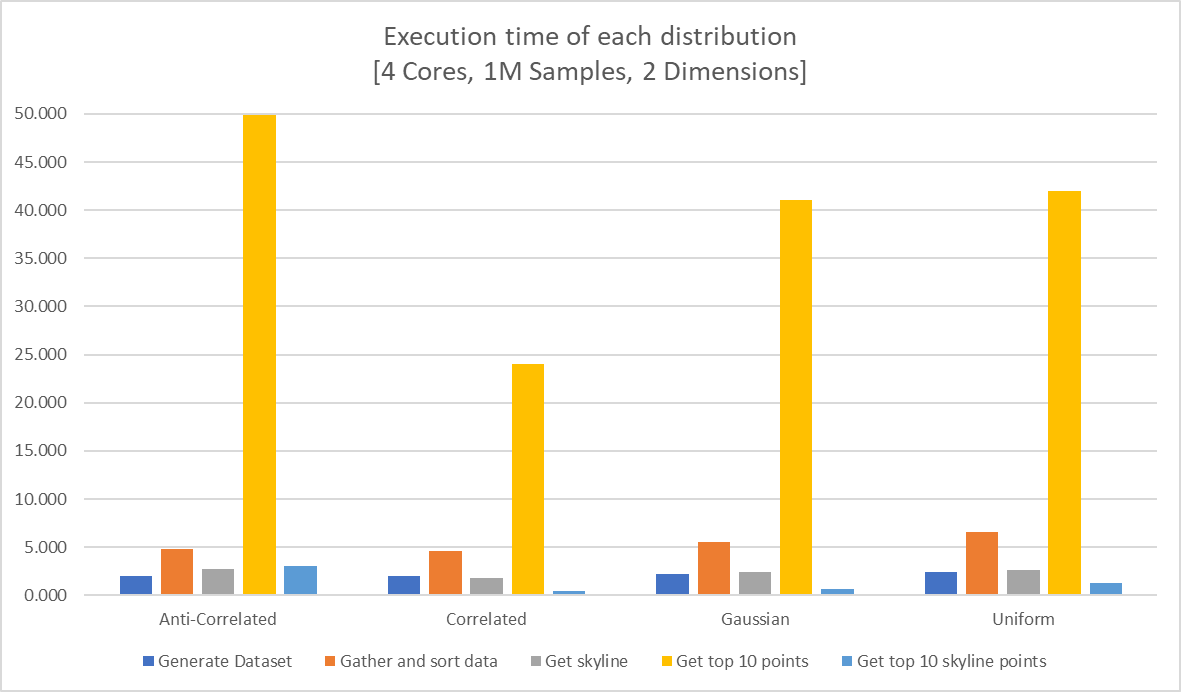
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PARAMETERS | | | | | | |
| Dimensions | 2 | | 3 | 4 | | |
| Dataset Size | 250k | 500k | 1M | 2M | 10M | 100M |
| *k* | 10 | | | 100 | | |

**Table 1:** Parameters

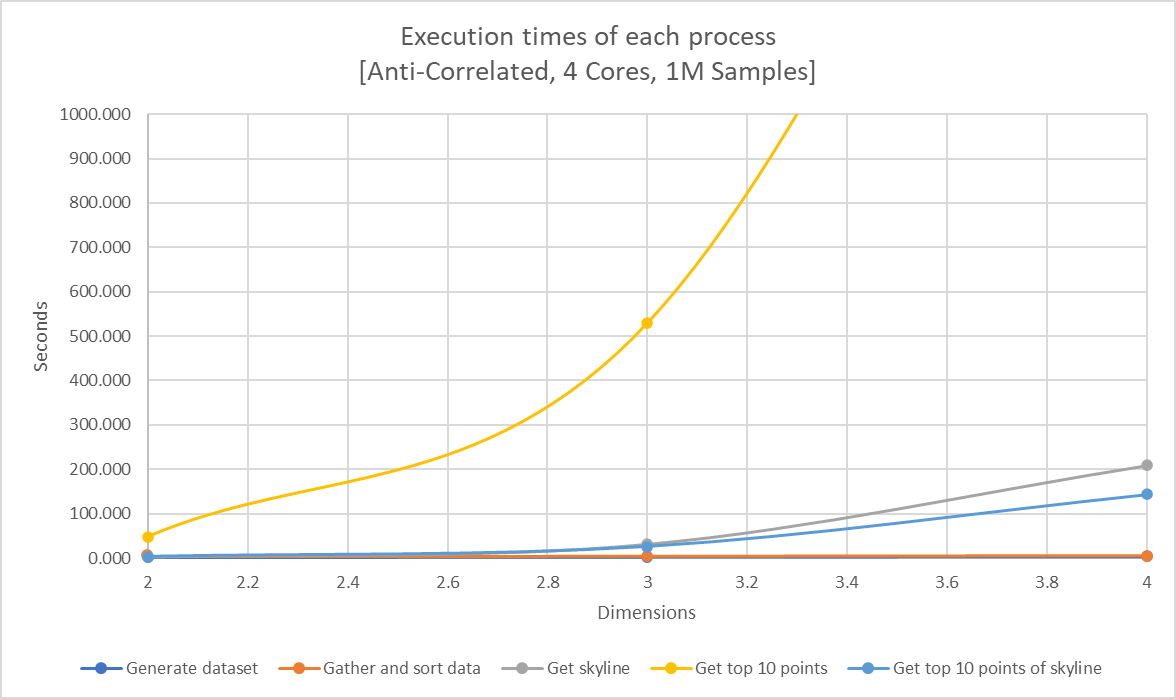
In order to describe the performance and scalability of our algorithms for different parameters, we chose anti-correlated distribution, because this distribution seems to be the most difficult one for all tasks.

3.1 Execution time for each distribution

We executed our spark script using datasets with 2 dimensions, 1 million samples and all the different distributions we are studding. As expected, the anti-correlated distribution is the most computational expensive one. We can also see that from the three tasks, the second one is the most expensive and with a big difference from the rest.

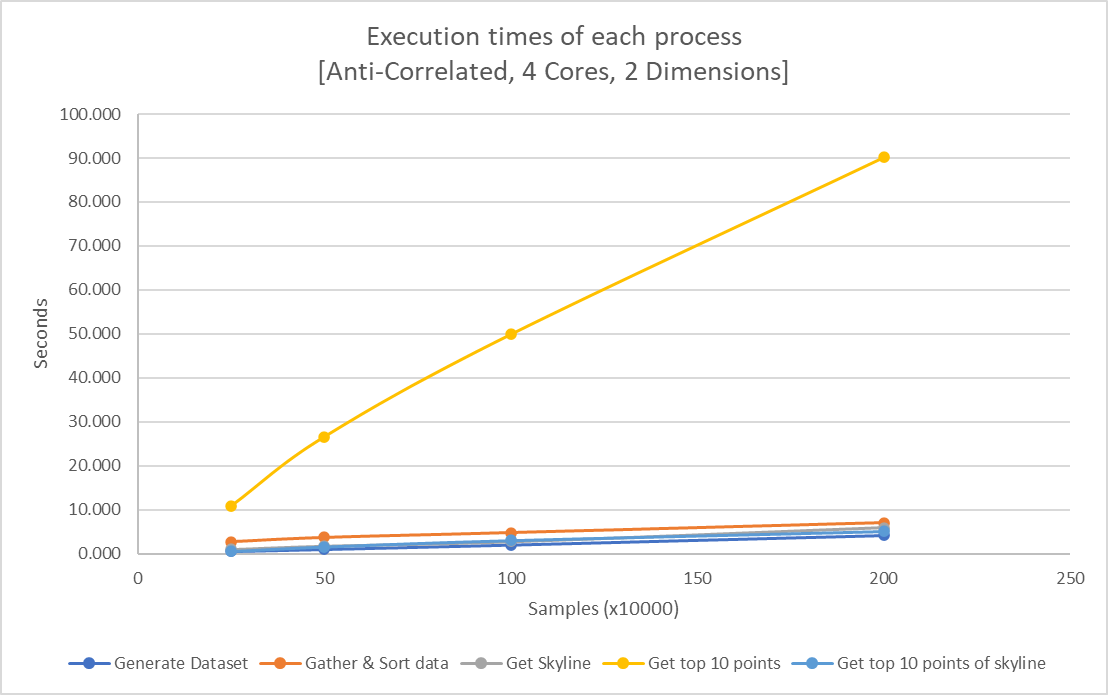
**Fig. 4:** Execution time in seconds of each task for the different distributions, using 4 cores and 1 million samples with 2 dimensions

3.2 Execution time VS Dimensionality

 Fig. 5: Execution time in seconds of each task for different sample dimensions, using 4 cores and 1 million samples. (The third point for the yellow line is out of the graph)

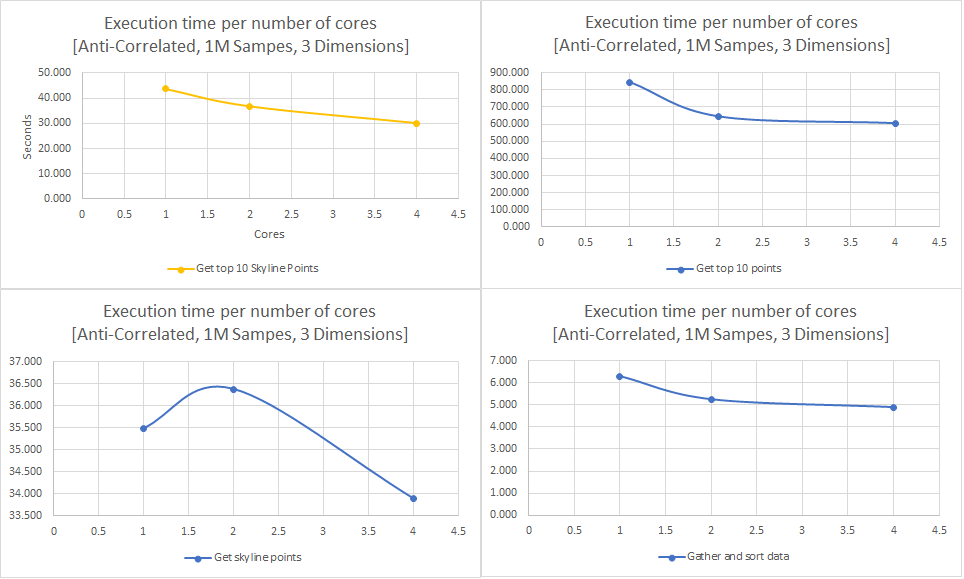
The increase in dimensionality has a great cost on computation time. The execution time increases exponentially, especially for task 2 (yellow line) which is the most complex of the three.

3.3 Execution time VS Sample size

We run our script for different dataset sizes with anti-correlated distribution.  Fig. 6: Execution time in seconds of each task for different sample sizes, using 4 cores and samples with 2 dimensions.

We choose to do this test with the anti-correlated distribution because it is the most computational heavy one and so we get the worst-case scenario results while for the rest distributions they can only be better. As we can see from the graph, the increase in the sample size seems to have a linear relation with the execution time.

3.4 Execution time VS Number of cores

FIG. 7: Execution time in seconds of each task for different number of cores using 1 million anti-correlated samples.

We executed our script on an anti-correlated dataset with 1 million samples using different number of scores in order to check if our implementation can scale. As we ca see from the graphs above, the execution time is getting reduced with the increase of number of cores which means that the solutions seem to scale.

4 CONCLUSION

In this paper, we introduced skyline and *top-k* dominating query, describing briefly the way they work, their advantages and how useful they are for data management and decision-making. Later, we presented our implementation for solving a case problem by calculating top-k points and skyline set. We also discussed the results and efficiency of our implementation.

As we said before, *top-k* dominating queries and skyline queries are important types of queries that are widely used in databases and data management systems. 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. Consequently, data are increasingly stored in a distributed way, therefore distributed query processing has become an important

and challenging problem. There are several challenges in dealing with *top-k* dominating queries and skyline queries, including the need to efficiently process large datasets, the need to handle complex dominance and preference relationships, and the need to support efficient updates and maintenance of the queries over time.

There is ongoing work in the field to address these challenges and improve the performance and capabilities of top-k dominating queries and skyline queries. This includes the development of new algorithms and data structures, as well as the use of specialized hardware and distributed systems to support the processing of large and complex datasets.

ACKNOWLEDGMENTS

This paper was built for educational purposes for the lesson “Technologies for Big Data Analytics” of the MSc program “Data & Web Science” of Department of Informatics at Aristotle University of Thessaloniki.

REFERENCES

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

[2] Hose, K., Vlachou, A. A survey of skyline processing in highly distributed environments. The VLDB Journal 21, 359–384 (2012). https://doi.org/10.1007/s00778-011-0246-6.

[3] 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.

[4] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive Skyline Computation in Database Systems. TODS, 30(1):41–82, 2005.

[5] Eleftherios Tiakas, A. Papadopoulos, and Y. Manolopoulos. 2015. Top-k Dominating Queries : a Survey.Conference Name:ACM Woodstock conference

[6] Yiu, M.L., Mamoulis, N. Multi-dimensional top-k dominating queries. The VLDB Journal 18, 695–718 (2009). https://doi.org/10.1007/s00778-008-0117-y

Conference Short Name:WOODSTOCK’18

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00