Scalable Processing of Dominance-Based Queries

Skyline, top-k Dominating and top-k Skyline Dominating Queries

ABSTRACT

The technological advancements led to massive data generation with the aim of providing optimal control and supervision to critical decision-making procedures. The data are described effectively with different features, increasing the data dimension size. The dimension size expands the complexity of decision-making and hinders data interpretation for users. Preference queries have been developed to ensure effective data management and optimal decision-making by specifying certain criteria like dominance score. This report implements 3 different algorithms as a decision support mechanism, 1) skyline query, 2) Top-k dominating query and 3) Top-k skyline query. This report seeks to calculate fast and accurate query results by exploiting Apache Spark with code written in Scala language. All algorithms are evaluated in terms of scalability, for different distributions, dataset sizes and several cores. The report indicates useful assumptions for scalable and efficient implementations of the examined algorithms.

KEYWORDS

Skyline queries, Top-k-domination, dominance score, Spark, Scala

1 INTRODUCTION

In the emerging landscape of database systems and query optimization, skyline-based queries have been proven a significant aspect of multi-criteria decision-making procedures. Their unique approach to identifying records in a database that are not "dominated" by any other record on all dimensions, yields results that are best at at least one characteristic.

The crux of skyline-based queries lies in their capability of filtering out non-optimal options and recommending only the "best" choices in a multi-dimensional space to the users. When a problem is multifaceted making a decision is sort of perplexed. That is why most of the time, there is not a simple single universal best selection. For instance, in a real estate scenario, prospective clients may desire to detect properties that establish the equilibrium among cost, location, size and amenities.

However, implementing optimized skyline-based queries is a particularly challenging application. Those challenges comprise high computational costs relating to big data processing and also large datasets handling methodologies. The need for efficient skyline computation without exhaustive all-to-all comparisons requires streamlined scalable algorithms.

[1] https://pandas.pydata.org/
[2] https://numpy.org/
[3] https://scikit-learn.org/
[4] https://spark.apache.org/

1.1 Skyline queries

Nowadays data are optimally described with multiple features or dimensions. The increased pace of data dimensions indicates the necessity of conducting a skyline analysis. Skyline analysis refers to the calculation of specific points that present interesting characteristics, which are related to optimal decision-making in a multi-criteria application. For example [14] examines a sports application, in which every data point represents an NBA player at a specific career game and each dimension includes different statistics like the number of points, rebounds, assists or fouls to identify the player with the best performance in a majority of aspects. Another application examines the Skyline analysis from a financial view to select the optimal hotel in terms of price and distance from the beach [3]. In short, skyline points express optimal solutions across different features, which are adjustable to user preferences and support the decision-making of various applications.

The definition of the skyline requires the definition of a dataset according to [5], which considers an initial dataset D with points in an n-dimensional space R^n. For a given point $p \in D$ where $p = (p.x1, p.x2, \dots, p.xn)$ and $\forall q \in D$, where $q = (q.x1, q.x2, \dots, q.xn)$. The p point is considered as a skyline point only if condition $p \leq q$ is satisfied for at least in one dimension. More simply, the p point must not be dominated by other points, meaning that p should be as good as q and strictly better in one dimension. For example point (1, 1) dominates point (1, 2). Figure 1 illustrates the skyline points p4, p1 and p3 for a two-dimensional dataset.

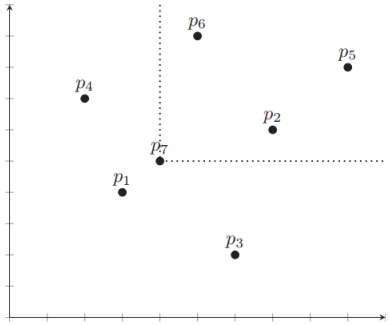

Figure 1: Skyline point for two dimensions datasets.

1.2 Top-k dominating query

The skyline queries provide points that are the optimal points for all dimensions or partially optimal in specific dimensions. However, the selection of the skyline points is not always a

feasible choice for the users due to unavailability or competition, taking into consideration the finite number of skyline data points. An extension of the skyline queries is the Top-k dominating query, in which the user defines a parameter k and the query returns k number of points with the greatest ranking function score to support the multi-criteria decision-making process. The calculation of the Top-k point requires the calculation of the skylines. The user should select a bigger number of k than the number of skylines otherwise a skyline query will be executed. Also, the combined calculation of skylines and Top-k dominating queries results in merging the advantages of the Top-k dominating queries and skyline queries without reflecting the drawbacks of each approach. More specifically the user can control the number of desirable important points in contrast with the skyline queries which cannot be defined by the user. It is worth noting that the Top-k ranking function must be carefully specified to calculate important points [5]. To eliminate the complexity of the selection of the ranking function most of the research studies determined the dominance score as the ranking function [1, 3, 5]. In addition, Top-k points reflect important points that are bounded in a specific subset of the skyline points. The examined subset is named the domination region. Figure 2 depicts the domination region of a random 2-D data point. Figure 3 depicts the Top-k 4 dominating query results for a 2-D dataset.

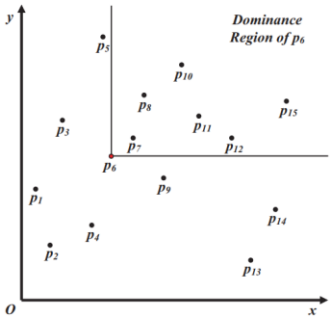

Figure 2: Dominance region of a 2-D data point.

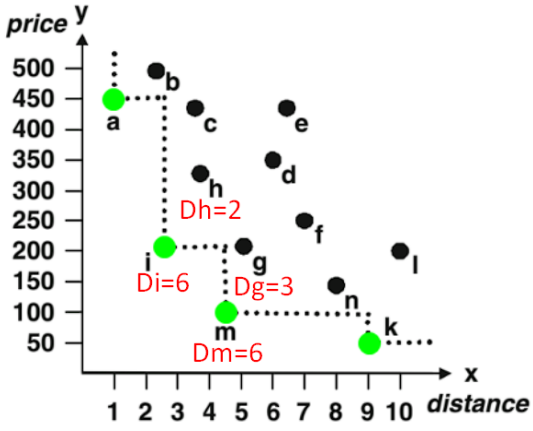

Figure 3: Top-4 dominant points for 2-D problem.

1.3 Top-k skyline dominating query

The rest of the paper is organized as follows: Section 2 describes the proposed algorithm for each task. Section 3 provides results for a variety of experiments of the implemented algorithm. In section 4 we discuss about the limitations and the potential fixes of the implemented algorithm.

2 IMPLEMENTATION

In this section, we are going to present the steps to which we adhere with the aim of tackling skyline, top-k dominating, and top-k skyline dominating queries. First and foremost, it was vital to create n-dimensional datasets of different cardinalities and distributions. That being said, Python programming language was utilized especially common libraries such as Pandas[1], NumPy[2] and scikit-learn[3] (Min-Max Scaling).

Concerning the combinations of dimensions, data cardinalities and distributions, we experiment with:

1. Dimensions: 2, 3, 4, 5, 6
2. Data Cardinalities: 25K, 100K, 250K, 500K, 1M, 2M, 10M
3. Distributions: anticorrelated, correlated, normal, uniform

We did not produce all possible conflations of the aforementioned parameters, but individually, each of them was used at least once in our experiments. Data values were universally normalized between [0, 1]. Figure 4 shows the visualization of the different distributions.

It is worth mentioning that in the case of anticorrelated distribution, we use equation (1) to ensure pair-wise anticorrelated relation between all data dimensions. n denotes the number of the

$$\Sigma = \begin{bmatrix} 1 & \rho & \cdots & \rho \\ \rho & 1 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & 1 \end{bmatrix} = (1-\rho)I_{(n)} + \rho e e^{\top} \quad (1)$$

with $\rho \in (-(n-1)^{-1}, 0)$ as a valid example.

dimensions and Σ is the equi-correlation matrix. Finally, we converted the equi-correlation matrix to a covariance matrix which was needed to use the multivariate_normal() method of NumPy library.

2.1 Task 1

In Task 1 the main purpose is to calculate the skyline set, that is all the points that are not dominated by any other point in the dataset. To do that, we took advantage of the All Local Skyline (ALS) methodology [15] for the distributed/parallel computation and the Sort-Filter Skyline (SFS) algorithm [12] for the calculation of the skyline.

The SFS algorithm comprises the listed steps below:

1. Sort data points based on the sum of the coordinates.

2. Add the first point of the sorted data points to the skyline set and remove it from the sorted data points set.
3. Discard all points that are dominated by the skyline of (2).
4. Go to (2) until the sorted data points set is empty.

To execute SFS in SPARK[4] we incorporated an ALS approach. The idea is that in each data partition, SFS is going to be applied individually. Then, the local skylines are to be returned to Driver and later, the universal skyline is calculated by applying again the SFS algorithm on the union of local skyline points.

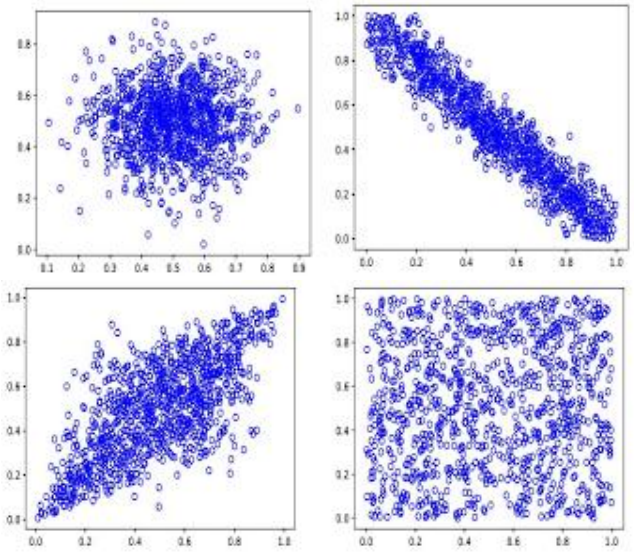

Figure 4: Different kinds of distributions included in the final experiments (top-left normal, top-right anticorrelated, bottom-left correlated, bottom-right uniform).

2.2 Task 2

Task 2 is the hardest among the three and it involves the calculation of the top-k dominating points, namely, the k first points with the highest dominance score. In this task, we adopted a Skyline-Based Top-k Dominating General Algorithm (STD) [1] that makes use of skyline query. STD consists of:

1. Calculate skyline points with the SFS (ALS) algorithm of Task 1.
2. Compute the skyline points dominance scores and sort them.
3. Move the point with the higher dominance score to the result buffer.
4. Calculate the skyline query in the dominance region of the point in (3).
5. Compute the dominance score of the skyline points in the dominance region of (4).
6. Add the skyline points of (4) and the respective dominance scores to the universal skyline points buffer.
7. Sort the buffer again.
8. Go to (3) until top-k points are returned.

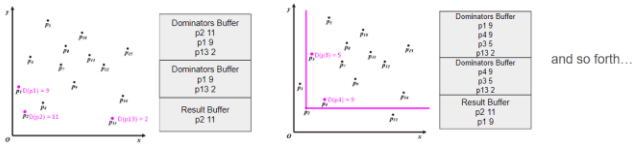

Figure 5: STD algorithm for an iteration.

The main idea here is that the skyline point whose dominance score is the highest will be always the top-1 dominator. Thus, iteratively we can find the top-k dominators. To calculate the dominance score of a point, we simply filter() and count() on data points set using the coordinates of the target point, whereas given (if it is needed) a new origin coordinates. The latter was applied to dominance calculation in dominance regions.

2.3 Task 3

In Task 3 the central goal is to recognize the top-k skyline dominators. That is the top-k dominators out of the skyline set. The steps we follow are:

1. Calculate skyline points with the SFS (ALS) algorithm of Task 1.
2. Compute the dominance scores of skyline points.
3. Sort them according to the dominance scores.
4. Return the first k points.

3 RESULTS

The results section of the conducted report seeks to present meaningful insights regarding scalability and performance through experimentation with different numbers of cores for various tasks, different distributions, a variety of data distribution cardinalities and a variable number of data dimensions. It is mandatory to declare that a validation procedure has been initiated to ensure the proposed algorithm results' accuracy. The validation procedure consists of a Brute Force Algorithm, written in a Python script. The rationale of the script is to make all possible comparisons among the data points. It is declared that the validation procedure took place for 2500 data points, tested for all distributions. The implemented algorithm presented the same results with the Brute Force Approach and it was assumed that the accuracy of the algorithm was ensured. Yet another significant issue refers to the anticorrelated distribution. Specifically, this report simulates anticorrelated relationships among all dimensions of data points, resulting in a huge computation cost. For example, for 1 million points our algorithm had been running for over 20 hours and still did not finish. Due to time limitations, the authors decided to experiment only with 100k data points up to 6 dimensions particularly for anticorrelated distribution. However, the maximum dataset size of anticorrelated points reaches one million data points only for 2 dimensions to extract useful insight for the corresponding distribution. The following subsections describe various experiments and conclude different insights.

3.1 Dimensionality investigation

This section investigates the dimensionality impact on different-size datasets. The dimensionality investigation is split for one million points for all distributions, except anticorrelated, from two to four dimensions. The other experiment was for 100K points with dimensions ranging from 2-D to 6-D for all distributions. Both experiments are conducted for four cores and all tasks. Figure 6 shows that Task 2 is the most computation-intensive one. Additionally, the execution time across different dimensions for uniform and normal distributions is increased exponentially in contrast with the correlation which follows a linear rate. Task 1 presents the same execution time regardless of the distribution. Task 3 increased exponentially with a small rate of change. Finally, Task 1 and Task 3 execution time are considered negligible compared to Task 2.

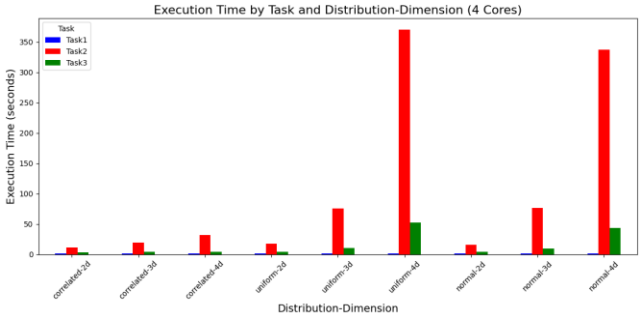

Figure 6: Experiments for one million points, 4 cores for all tasks and distributions except anticorrelated.

The second experiment tested the proposed algorithm for up to 6 dimensions for 100k data points. It is apparent that the anticorrelated distribution is the most time-consuming. One important observation is that as dimension size increases Task 2 execution time is very close to the execution time of Task 3 which is reasonable due to the increased number of skyline points in the anticorrelated distribution. As the number of skyline points increased then the execution time increased proportionally to identify the corresponding dominance scores. Figure 7 presents the results of the second experiment.

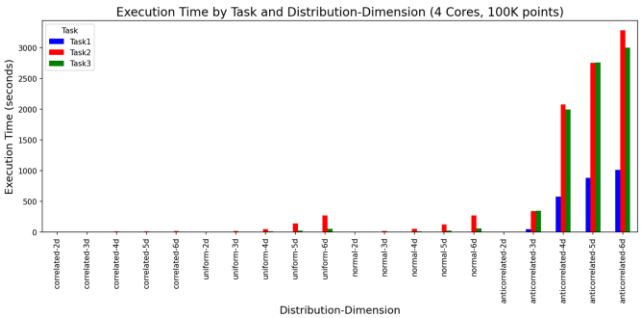

Figure 7: Experiment for 100k data points, 4 cores for all distributions.

3.2 Scalability experiments

This section provides two major experiments regarding the number of cores to illustrate the scalability.

For the first experiment, all distributions were generated for two and three dimensions, except anticorrelated. A qualitative comparison of one million points for correlated, uniform, and normal distribution is being conducted. Figure 8 proves that with an increase in the number of cores, the execution time is decreased so our algorithm is scalable. Scalability is more visible in 3 dimensions for uniform and normal distribution where the execution time for two cores starts at 120 sec approximately and with four cores falls to 80 sec for both distributions. It is worth noting that scalability can be observed clearly in 3 dimensions.

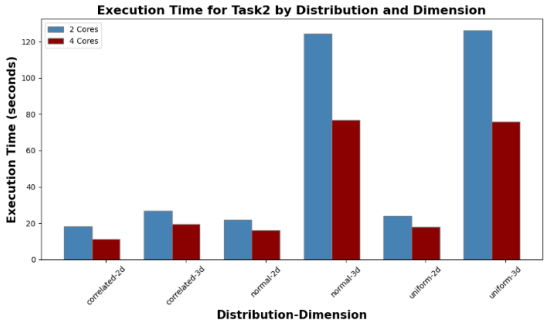

Figure 8: Experiments with comparisons of 2 cores vs 4 cores with size one million except anti-correlated for 2-D and 3-D.

The second experiment generates results for ten million data points, including all distributions, two dimensions both for two and four cores. Once again, the scalability is proven and it is observable in the anticorrelated distribution which ranges from 330 seconds for two cores to 200 seconds for 4 cores. Similar to the first experiment, the correlated distribution is the least computationally intensive. Figure 9 illustrates the results of the second experiment.

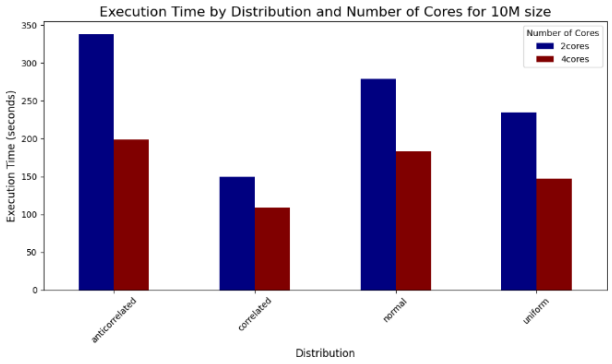

Figure 9: 2 cores vs 4 cores with size 10 million except with all distributions and 2-D.

3.3 Impact of the Dataset Size

Following, more experiments were carried out to identify for each distribution the impact of a number of the data points, which were implemented for two dimensions. It is worth noting that Task 2 is

the most computation-intensive task followed by Task 3 and Task1.

Similar to the increased dimensions the Task 2 execution time increases exponentially at a high rate. Task 3 increased exponentially with a lower rate that is better observed in the anticorrelated distribution. Task 1 seems to be independent regarding the dimension although it presents a slight increase in the one million number of points. Figure 10 presents the results for the correlated distribution. Figure 11 shows the results for the uniform distribution. Figure 12 depicts the results for the anticorrelated distribution and Figure 13 pictures the results for the normal distribution.

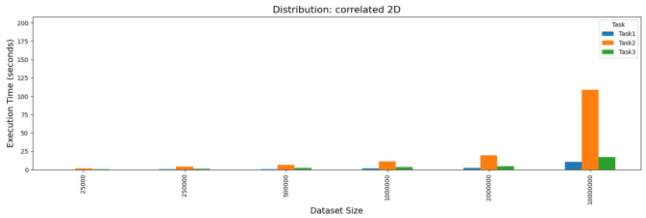

Figure 10: Experiment for an increased number of dataset sizes for correlated distribution.

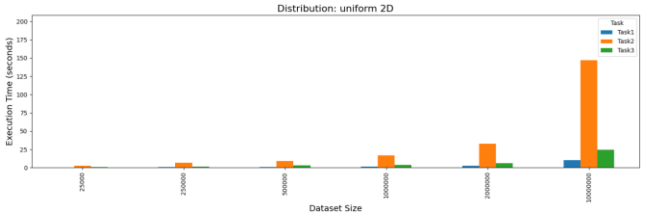

Figure 11: Experiment for an increased number of dataset sizes for uniform distribution.

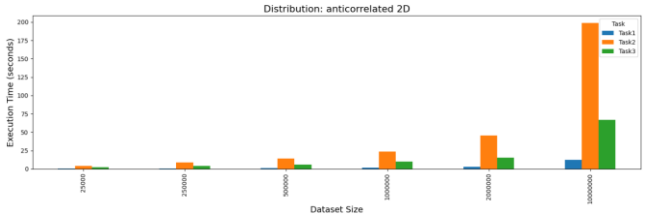

Figure 12: Experiment for an increased number of dataset sizes for anti-correlated.

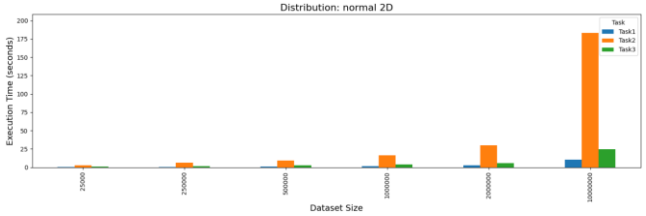

Figure 13: Experiment for an increased number of dataset sizes for normal distribution.

The final diagram depicts the change in execution time for different distributions for Task 2 as the number of data points increases. It is obvious that the anti-correlated presents the biggest execution time followed by normal, uniform and correlated. Figure 14 depicts the results for the mutable size of data points for all distributions. An increase in the data points size triggers a linear growth of the execution time.

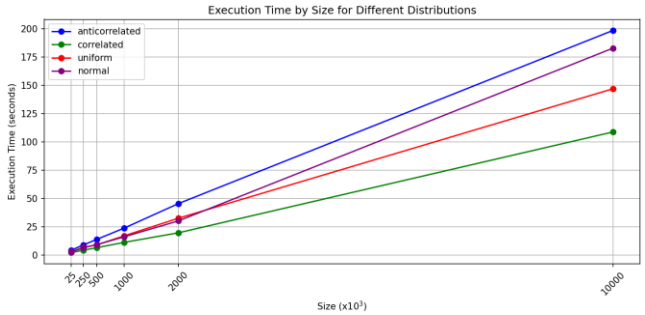

Figure 14: Execution time per distribution for a mutable number of data points for Task 2.

4 DISCUSSION

In this report, we begin by describing skyline queries and their usefulness concerning decision-making challenges. Then, a brief introduction is presented for the three main skyline-based queries which were the prime cases that were investigated in this work. Finally, our approaches to tackle the three problems in conjunction with the results are mentioned and analyzed thoroughly.

Taking everything into account, it is overt that Task 2 is the most time-consuming and calculation-intensive. It consists of iterative skyline queries and dominance score calculations. Specifically for the anticorrelated distribution, when the number of dimensions is higher than 2, the count of skyline points surges. This is the corollary of the all-to-all anticorrelated relation between each pair of dimensions.

Another concern emerges due to the large count of skyline points in the anticorrelated distribution. We observed that a vast proportion of the time consumed to calculate Task 1, 2 and 3 for dimensions greater than 2, was wasted on the collect() function. An alternative approach could be to regard the union set of local skyline queries as a new RDD and try applying transformations to it. However, the practicality of the tasks in which the majority of the points are included in the skyline set is dubious.
Last but not least, the experiments were carried out on a commonplace personal computer having 4 physical cores and 4 logical processors (threads). It would be worthwhile to observe how our algorithms are executed utilizing more cores in parallel.

ACKNOWLEDGMENTS

The work presented in this paper constitutes an educational endeavor to address skyline-based query challenges using Scala and Spark on behalf of the subject “Technologies for Big Data Analytics” of the MSc program “Data & Web Science” of the

Department of Informatics at the Aristotle University of Thessaloniki.

REFERENCES

[1] Tiakas, Eleftherios, Apostolos N. Papadopoulos, and Yannis Manolopoulos. "Top-k Dominating Queries." (2007).

[2] Amagata, Daichi, et al. "Efficient processing of top-k dominating queries in distributed environments." World Wide Web 19 (2016): 545-577.

[3] Borzsony, Stephan, Donald Kossmann, and Konrad Stocker. "The skyline operator." Proceedings 17th international conference on data engineering. IEEE, 2001.

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

[5] Yiu, Man Lung, and Nikos Mamoulis. "Multi-dimensional top-k dominating queries." The VLDB Journal 18 (2009): 695-718.

[6] JIA, MOXIN. "Grid and angular based partitioning skyline and top-k skyline queries with spark." (2022).

[7] Grasmann, Lukas, Reinhard Pichler, and Alexander Selzer. "Integration of Skyline Queries into Spark SQL." arXiv preprint arXiv:2210.03718 (2022).

[8] Pinari, Etion. "Parallel Implementations of the Skyline Query using PySpark." (2022).

[9] Papanikolaou, Ioanna. "Distributed algorithms for skyline computation using apache spark." (2020).

[10] Papadias, Dimitris, et al. "An optimal and progressive algorithm for skyline queries." Proceedings of the 2003 ACM SIGMOD international conference on Management of data. 2003.

[11] Παπαρίδης, Κωνσταντίνος. Παράλληλη και Κατανεμημένη Επεξεργασία Ερωτημάτων Κορυφογραμμής στο σύστημα Spark. Diss. Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, 2016.

[12] Bartolini, Ilaria, Paolo Ciaccia, and Marco Patella. "Efficient sort-based skyline evaluation." ACM Transactions on Database Systems (TODS) 33.4 (2008): 1-49.

[13] Papadias, Dimitris, et al. "Progressive skyline computation in database systems." ACM Transactions on Database Systems (TODS) 30.1 (2005): 41-82.

[14] Jian Pei, Bin Jiang, Xuemin Lin, Yidong Yuan, “Probabilistic skylines on Uncertain Data” , VLDB 23-28, 2007, Vienna, Austria.

[15] Vlachou, Akrivi, Christos Doulkeridis, Yannis Kotidis, and Michalis Vazirgiannis. "Skypeer: Efficient subspace skyline computation over distributed data." In 2007 IEEE 23rd International Conference on Data Engineering, pp. 416-425. IEEE, 2006.