Scalable Processing of Dominance-Based Queries

George Matlis
Department of Computer Science
Aristotle University of Thessaloniki
Thessaloniki, Greece
gmatl@csd.auth.gr

Konstantinos Tsintzas
Department of Computer Science
Aristotle University of Thessaloniki
Thessaloniki, Greece
tsintzask@csd.auth.gr

***Abstract*—This application delves into the domain of dominance-based queries, crucial in complex decision support systems, and focuses on scalable processing methodologies. This study investigates the utilization of the Scala programming language, and the Apache Spark engine for the distributed implementation of dominance-based query processing. These technologies are pivotal in addressing the challenges posed by large-scale datasets, aiming to efficiently identify superior alternatives within vast data repositories.**

***Index Terms*—Scala, Apache Spark, Skyline, Scalable Processing, Dominance-Based Queries**

I. Introduction

The skyline operator represents a powerful concept in multi-criteria decision-making within the field of database systems. [1] It operates on multi-dimensional data and identifies a subset of non-dominated points from a very large dataset. A skyline set is comprised of data points that are not outperformed by any other point in all dimensions, making them the most significant and non-dominated choices for decision-making.

For d-dimensional data, the concept of dominance is fundamental to the skyline computation. Dominance establishes the relationships between data points across multiple dimensions. A point A is said to dominate point B if and only if A is equal to or better than B in all dimensions and strictly better in at least one dimension. For example, in a two-dimensional space, if point A has a greater or equal value in both dimensions compared to point B, and strictly greater in at least one dimension, then point A dominates point B.

The skyline operator leverages the concept of dominance in d-dimensional space to efficiently identify points that are not dominated by any other, hence revealing a concise yet comprehensive set of optimal solutions for decision-making and analysis in multi-dimensional datasets.

The three tasks of this application are to:

1) Find the skyline set (the set of d-dimensional points that are not dominated by any other point in the dataset).
2) Find the top-k d-dimensional points with the highest dominance score. The dominance score is the number of points a point A dominates.
3) Find the top-k d-dimensional points with the highest dominance score from the skyline set.

We aim to solve the aforementioned tasks by using Scala as the programming language, along with the capabilities of Apache Spark in a distributed computing environment.

II. Dataset

For our application, we have generated four distributions for any arbitrary number of dimensions:

1) Normal distribution
2) Uniform distribution
3) Correlated distribution
4) Anti-correlated distribution

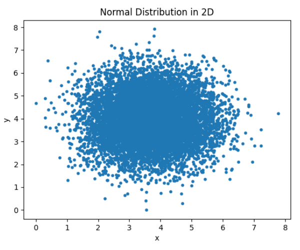
We have constrained the range of the distribution to encompass only positive values, ensuring that the d-dimensional points exclusively maintain positive coordinates. This choice is motivated by our use of the Euclidean distance as a scoring function in our experiments. This function assigns to each point the distance from its location to the origin of the axes. The calculated distance serves as a measure of how far a point is positioned from the origin, facilitating the sorting of points and theoretically simplifying the implementation of the three tasks. The four distributions are shown in fig. 1

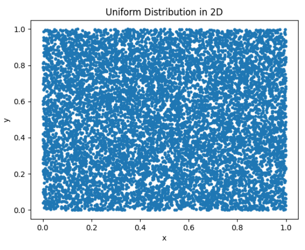
III. Methodology

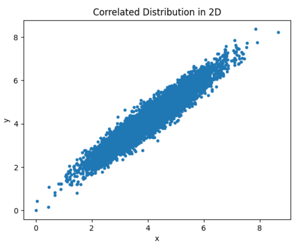
A. Task 1

For the first task, we generate a dataset consisting of 100,000 points for two, four, and six dimensions of the four distributions outlined in Section II. Subsequently, we partition the dataset based on the computational capacity, and finally, to compute the skyline set, we implement the following steps (see also Fig. 2):

1) We split the comma separated coordinate values present in the text file. The text file is essentially the dataset that includes all the points.
2) We convert each coordinate to a numerical value.
3) We create an array consisting of the coordinates of a point.
4) We execute a baseline skyline algorithm locally in each worker that takes the points as input and outputs the local skyline set. The algorithm removes all the dominated points leaving only those that are not dominated by any other point.
5) We collect the local skyline set from each worker. This process happens in the driver.


(a)


(b)


(c)

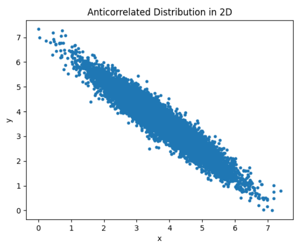

(d)

Fig. 1: Visualization of the four distributions in 2D

6) We execute the same baseline skyline algorithm to get the global skyline set.

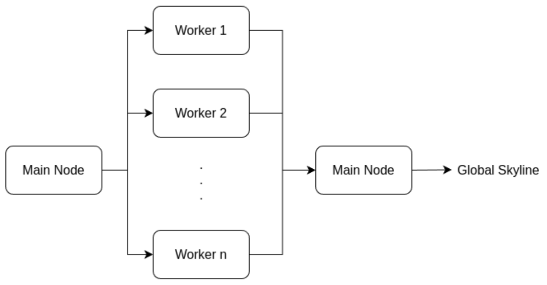


Fig. 2: Skyline Calculation Model

The results of the implementation of the aforementioned steps are delineated in Table I.

Upon reviewing Table I, a substantial difference in computational time is evident between two workers and four workers. However, the increase in computational time when transitioning from four workers to six workers is minimal. This marginal difference can be attributed to the computational capacity of the machine used and the dataset size. We hypothesize that with a larger dataset and more capable machines, the time difference between experiments employing four and six workers would become significant.

Analyzing the algorithm's performance across distributions, a significant increase in computation time is observed for the six dimensions in the case of the Normal and Uniform distributions. This is anticipated due to the generation of numerous local skyline points, leading to an extended computation time for the global skyline points. In contrast, the correlated distribution, being simpler, results in local skyline points that are comparatively easier to compute compared to the other distributions. Surprisingly, the algorithm's performance on the anticorrelated distribution differs from our initial expectations given its complex shape and possible generation of numerous global skyline points. This unexpected outcome is attributed to the fact that each global skyline point in the anticorrelated distribution dominates a relatively smaller subset of points compared to the other distributions. Consequently, this leads to significantly reduced dominance regions. In contrast, global skyline points in the correlated distribution, despite their scarcity, are likely to dominate a larger portion, if not all, of the points, resulting in comparatively larger dominance regions.

TABLE I: Skyline Calculation Results for 100,000 points

Distribution	**Dim**	**2W**	**4W**	**6W**
Normal	2	0.82	0.39	0.47
	4	10.35	3.33	1.99
	6	133.62	74.52	73.05
Uniform	2	5.55	2.08	1.58
	4	6.35	4.11	3.78
	6	107.05	75.80	69.07
Correlated	2	0.08	0.08	0.07
	4	0.12	0.09	0.09
	6	0.14	0.11	0.13
Anti-Correlated	2	11.57	3.75	2.36
	4	8.82	3.27	2.12
	6	22.84	9.94	8.53

One potential concern associated with the outlined approach is the dimensionality of the points and their distribution. In the event of an increase in the dimensionality of the points, particularly in conjunction with an anti-correlated distribution, a proliferation of potential local skyline points may ensue, impacting the performance of the driver machine. To mitigate this issue, we could merge the local skyline sets of selected workers (rather than all workers) into a single worker, and producing a new local skyline set. Subsequently, the results are collected and relayed to the driver. This strategy reduces the number of potential local skyline points the driver needs to process, thereby preventing a significant degradation in performance. By selectively combining the results from multiple workers, this strategy aims to strike a balance between computational efficiency and the accurate identification of local skyline points, optimizing the overall performance of the system.

B. *Task 2*

For the second task, we generate two datasets, consisting of 1,000 and 10,000 points respectively for two, four, and six dimensions, for each of the four distributions described in Section II. No special partitioning is done for this task on the worker level. Instead, we work by constructing a virtual grid to organize the cells into local groups. The steps for creating the grid are listed below:

1) Receive the dataset and the requested number of points per cell as input.

2) Find the maximum and minimum values for each dimension.
3) Calculate the number of cells along each dimension, with the requested number of points per cell as the maximum. This assumes a uniform distribution of points.
4) Construct the uniform grid, dividing the space of our dataset into separate cells. Cells closer to (0,0) are more likely to contain dominant points.

Next, we leverage the grid and a greedy strategy to locate candidate dominant points to reduce the number of dominance comparisons we need to make, execute what comparisons still must be done, sort our calculated points in descending order based on dominance, and return the first *k* points.

1) Count the number of points in each grid cell based on each point's coordinates, in a distributed fashion.
2) Select all cells closest to each axis, and consider the points within them as candidate dominant points. If there are less than *k* points in the candidate set, additionally select all adjacent cells. Repeat this process until at least *k* candidate points are selected.
3) Construct the Cartesian product of candidate points and all points that are in cells not dominated by the respective candidates.
4) Perform distributed dominance comparisons using the Cartesian product.
5) Sum the dominance scores of identical points based on coordinates, and add the number of points in their respective dominated cells to each candidate point.
6) Sort the candidates in descending order based on dominance score, and return the top *k* candidates as the result.

The time performance of the above process is shown in in Tables II and II. Between the normal, uniform, and correlated distributions, the results are as expected: The correlated distribution performs the fewest dominance comparisons, followed by the normal distribution, and the uniform distribution. The algorithm performs quite fast for the anti-correlated distribution, but this is due to a failure of the greedy method to correctly select candidate points. This is elaborated upon below.

Subsequent testing has also determined that grid construction and point tallying takes approximately 1 to 2 seconds, irrespective of dimensionality and number of workers; it is a very fast process. The bulk of time consumption is for dominance comparisons.

A few caveats and challenges present themselves regarding this method of calculating dominance scores. First and foremost, the greedy method utilized in selecting candidate dominant points fails to account for irregular distributions such as the anti-correlated distribution, and in such cases, selects unsuitable candidate points. This results in receiving a false output for the top dominant points.

Two possible avenues of remedying this exist: selecting all points from cells containing skyline points, or using an alternative heuristic to locate promising candidates, such as the sum of a point's coordinates. Due to time constraints, testing these was not feasible, but we hypothesize both would result in an expected slowdown. Selecting all points from cells containing skyline points would induce greater slowdown with a bigger skyline, while also adding the cost of the skyline computation to the overall time. On the other hand, using the sum of a point's coordinates would require selecting exponentially more candidate points in higher dimensions to ensure accuracy.

It would also be possible to sample the dataset to determine its distribution, then dynamically select an appropriate method of determining candidate dominant points.

Another caveat is that during the sum step of the dominance calculation, it is assumed that points with identical positions are the same object, which might not always be the case. This could be solved by incorporating unique point IDs into the dataset, and utilizing those to sum the dominance scores.

One parameter to watch out for is the cell size. Should it be too large, higher-dimensional datasets will have too few cells, resulting in minimal or no time gain compared to an all-to-all calculation, as few to none dominance comparisons may be skipped.

TABLE II: Dominance Query Calculation Results (For top 10 points out of 1k)

Distribution	**Dim**	**2W**	**4W**	**6W**
Normal	2	1.08	0.79	0.99
	4	4.84	3.16	3.07
	6	6.59	3.92	4
Uniform	2	3.42	2.42	2.7
	4	5.11	3.62	3.52
	6	7.06	4.59	4.76
Correlated	2	0.67	0.51	0.52
	4	3.34	2.01	2.13
	6	5.16	3.19	3.26
Anti-C	2	0.75	0.5	0.52
	4	5.14	2.68	2.78
	6	6.58	3.78	3.89

TABLE III: Dominance Query Calculation Results (For top 10 points out of 10k)

Distribution	**Dim**	**2W**	**4W**	**6W**
Normal	2	13.07	7.52	7.06
	4	244.96	128.91	173.95
	6	376.96	200.26	196.26
Uniform	2	99.35	52.7	52.41
	4	269.73	142.34	142
	6	415.39	224.87	211.79
Correlated	2	5.97	3.42	3.24
	4	173.16	94.45	92.59
	6	253.85	136.5	128.91
Anti-C	2	8.34	4.44	4.27
	4	261.37	139.85	130.89
	6	372.12	208.06	193.45

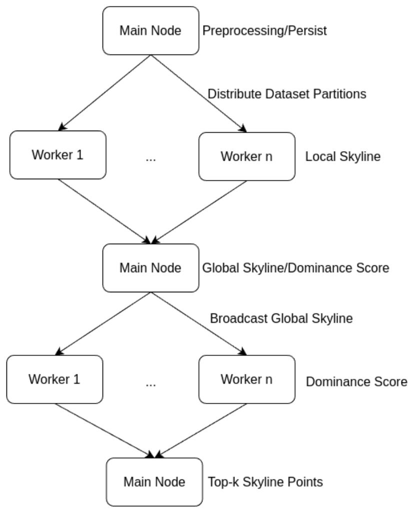


Fig. 3: Top-k Skyline Calculation Model

C. Task 3

The third task represents a natural extension of the first task, featuring a slight modification to the baseline skyline algorithm. Prior to executing the baseline skyline algorithm, we employ the *persist* functionality to retain the Resilient Distributed Dataset (RDD) generated during the pre-processing stage—comprising the first three steps detailed in Task 1. This modification involves augmenting the baseline algorithm to store the length of each partition (indicating the number of points each worker processes) and the indices of local skyline points locally on each worker. In the computation of the global skyline set, we leverage the original baseline skyline algorithm, incorporating an additional step wherein we calculate the dominance score of the final global skyline set. After the execution of the first task, we implement the following steps (see also Fig. 3):

1) We broadcast the global skyline set.
2) In each worker, we load the indices of the local skyline points and the length of the partition. Subsequently, we construct a list starting from 0 and ending at *partition_length*, encompassing all indices except those corresponding to the local skyline points. This process effectively generates the indices of all the dominated points within the partition. With this information in hand, we proceed to the final step.
3) We compute the dominance score for each global skyline point by comparing its coordinates with the coordinates of the dominated points.

The results of the implementation of the aforementioned steps are delineated in Table IV

The findings presented in Table IV align with those discussed in the first task, given the natural extension of the third task from the initial analysis. Notably, the unanticipated performance in the anticorrelated distribution persists. Acknowledging the intricacies of the third task, we employed 50,000 points for each distribution to ensure robust results.

TABLE IV: Top-k Skyline Calculation Results for 50,000

Distribution	Dim	2W	4W	6W
Normal	2	0.86	0.34	0.20
	4	3.23	1.42	0.78
	6	62.24	36.26	33.24
Uniform	2	1.35	1.61	1.98
	4	2.59	1.73	1.56
	6	45.62	34.16	26.93
Correlated	2	0.08	0.08	0.08
	4	0.10	0.09	0.08
	6	0.13	0.09	0.09
Anti-C	2	3.38	1.45	0.90
	4	3.29	1.59	0.83
	6	11.42	4.97	4.03

Beyond the concern outlined in Task 1, an additional issue with this approach relates to the computational burden placed on workers during the calculation of the dominance score for the global skyline set. This challenge can be addressed by incorporating heuristics, such as employing a predefined threshold for the Euclidean distance to identify and exclude certain dominated points. Leveraging the assumption that dominating points tend to be closer to the origin of the axes, we streamline the process by sorting the points and flagging those that surpass a specified threshold. This heuristic offers an approximate solution to the task, potentially enhancing computational efficiency compared to the original approach. It is essential to emphasize that the efficacy of this theoretical approximation is contingent upon the characteristics of the distribution used.

IV. Conclusions

Dominance-based queries are a complex field of research, and have found many useful applications too. In this work, we focused on methods to execute such queries efficiently in a distributed context, where the data is stored in partitions. We explored distributed ways to compute skylines, and to calculate dominance scores. Our methods, though they faced challenges, were nonetheless able to effectively leverage the nature of distributed systems to achieve better performance with more executors.

V. Future Work

For potential further enhancements to these methods, it would be beneficial to delve deeper into understanding the algorithm's performance on the anticorrelated distribution in both Task 1 and 3, considering its characteristics and complex shape. It would also be advisable to explore the directions discussed regarding the improvement of the dominance calculation task. Additionally, exploring the application of further heuristics, such as leveraging the Euclidean distance or other metrics, could offer opportunities to enhance the implementation and reduce computational time of the algorithms. Finally, more robust experiments could be conducted by leveraging larger datasets and more powerful computing resources.

References

[1] E. Tiakas, A. N. Papadopoulos, and Y. Manolopoulos, “Top-k dominating queries,” 2007.

Appendix

The full code for this project can be found in the following GitHub repository: https://github.com/GeorgeM2000/Scalable-Processing-of-Dominance-Based-Queries