POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

EXECUTIVE SUMMARY OF THE THESIS

Computation of Flexible Skylines in a distributed environment

TESI MAGISTRALE IN COMPUTER SCIENCE AND ENGINEERING – INGEGNERIA INFORMATICA

AUTHOR: EMILIO DE LORENZIS

ADVISOR: DAVIDE MARTINENGHI

ACADEMIC YEAR: 2021-2022

1. Introduction

In recent years, as big data has grown, there has been an attempt to find a way to search within ever larger data sets for data that might be of interest and to which one should pay special attention. Skyline queries [1] provide an efficient and effective method for selecting a subset of data from large datasets. These queries selects tuples that are not dominated by any other tuple, yielding a set of data points that are considered more interesting. We have that a tuple t dominates another tuple s if and only if t is not worse in any attribute than s and strictly better in at least one. Skylines offer a global view of data that may be of interest, but they do not account for user preferences as they treat all attributes at the same level. In addition, skyline sets may contain an excessive number of tuples, which may not provide meaningful insights for decision-making. To address these limitations, this thesis also explores the concept of Flexible Skylines [2]. Flexible Skylines incorporate constraints on attributes, allowing for importance differentiation between attributes. The concept of F-dominance was introduced for this purpose: A tuple t F-dominates another tuple s if and only if t is always better than or equal to s according to all scoring functions in F. Flexible skylines thus return a subset of the skylines and are divided into 2 operators: ND which returns the subset of non F-dominated tuples and PO which returns a subset of ND representing all tuples that are potentially optimal with respect to a scoring function in F. An advantage of using Flexible skylines (F-skylines) is that they can potentially reduce the cardinality of the resulting tuples compared to traditional Skylines. This is because F-skylines incorporate constraints on attributes, allowing for greater filtering of the data. When the constraints become tighter, the resulting set of tuples gets smaller. In this thesis, several algorithms will be presented for computing Skylines and Flexible Skylines. It is important for us to introduce parallel frameworks to try to divide the computation of these sets into several partitions that will be executed in parallel, to try to speed up the execution time of this algorithm w.r.t. the centralized version. Each partition will have a part of the dataset and will return a local skyline that will be equal to the skyline set of that subset of points and not all the dataset. Once all the local skylines have been computed, they will merge and there will be a sequential phase in which we will find the global skyline from the local skylines found from each

partition. Our important goal is to find the best possible partitioning in order to have local skylines as small as possible and thus reduce the final sequential phase, which will be the slowest. Later in this thesis we will present improvements to these types of partitioning by performing an initial filtering phase, thus decreasing the load on the parallel part, and then we will see a method to eliminate the sequential phase totally. One aim of this thesis will be to implement the algorithms in the parallel versions with all the types of partitioning widely used for skyline queries and apply them to our specific case of the Flexible Skyline computation using the PySpark framework [4], and to verify which algorithm performs best and what conditions makes it perform the best.

1.1. Thesis contributions

The main contributions of this thesis are:

1. Pre-existing algorithms for skyline computation and Flexible Skyline operators are introduced and tested using Python as the programming environment. Next, the Spark framework, used to parallelize the computation of these algorithms, is introduced.
2. Take some partitioning algorithms from the literature applied so far only for Skyline computation, and readapt them for the computation of the Flexible Skyline ND and PO operators.
3. Introduce some improvements to these parallel algorithms, some taken from the literature applied to the Skyline computation, others introduced for the first time in this thesis. Another contribution will be to take this technique, and readapt it to the computation of the ND and PO sets with some improvements.
4. Introduce a parallel algorithm without a sequential part that is good for the computation of the two Flexible Skyline operators ND and PO.
5. Implement all the algorithms using the Pyspark framework and derive an experimental analysis in which we will test the efficiency of the various algorithms and find the optimal setting to make them work best using synthetically created datasets.

2. Preliminaries

2.1. Skyline query

The skyline query was first introduced in [1] to find all the best tuples in a database. It is based on the concept of dominance, in fact a skyline query returns all tuples (points) that are not dominated within a dataset. In this thesis we consider smaller values as better, but it is only a convention we use here, the opposite could be used. Let us first introduce the definition of dominance between tuples.

Definition 2.1: Given two tuples t, u $\in \mathbb{R}^d$ belonging to the same dataset S, t dominates u, written $t \prec u$, if and only if t is not worse than u in all dimensions and better in at least one. Equivalently:

$$t \prec u \leftrightarrow \begin{cases} \forall i \in [1,d] \rightarrow t[i] \leq u[i] \\ \exists j \in [1,d] \rightarrow t[j] < u[j] \end{cases}$$

Thus we have that the skyline set of a Dataset S, denoted by SKY(s) is equal to:

$$SKY(s) = \{t \in s | \nexists u \in s.\ u \prec t\}$$

2.2. Flexible Skyline

From now on, we will introduce another type of dominance between tuples that takes scoring functions into account. A scoring function f is a function that takes a tuple with non-negative real values for attributes and returns a non-negative real value representing the score.

Definition 2.3: (F-dominance) Let F be a non-empty set of monotone scoring functions and t, u tuples belonging to a dataset S, with $t \neq u$. We have that t F-dominates u, written as $t \prec_F u$, if:

$$\forall f \in F \rightarrow f(t) \leq f(u)$$

2.2.1 Non-dominated Flexible Skyline (ND)

We now introduce the first Flexible Skyline operator, called non-dominated Flexible Skyline (ND), which is the set of all tuples that are not F-dominated within a dataset.

Definition 2.5: The set of non-dominated Flexible Skyline (ND) of a dataset s with respect to a set of monotone scoring functions F ⊆ MF, is defined as:

$$ND(s;F) = \{t \in s \mid \nexists\, u \in s.\ u \prec_F t\}$$

There are two main strategies for computing ND. In the first one we have to solve a different LP problem for each of the F-dominance Tests, the

second strategy instead, involves computing the dominance region of each point and eliminating all points that are part of at least one of these regions.

2.2.2 Potentially Optimal Flexible Skyline (PO)

The second operator, called Potentially Optimal Flexible Skyline (PO), returns a set of tuples considered optimal (the best) according to some monotone scoring functions F.

Definition 2.7: The set of Potentially Optimal Flexible Skyline (PO) of a dataset s with respect to a set of monotone scoring functions F ⊆ MF, is defined as:

$$PO(s;F) = \{t \in s \mid \exists f \in F . \forall u \in s.\ u \neq t \rightarrow f(t) < f(s)\}$$

We have that a tuple t that is potentially optimal is certainly also ND, but the fact that it is not F-dominated is only a necessary condition for t to be potentially optimal and not sufficient. There are two ways to find the PO set: start from the whole dataset and find the set in one step or find the ND set first and then compute PO from that. In both cases, we have to solve some LP and there are two strategies: solve the Primal or the Dual PO test.

3. Sequential Algorithms

3.1. Skyline Algorithms

The skyline algorithms seen in this thesis are Block Nested Loop (BNL), SFS and SaLSa. For the experimental results, we will only use SFS, which is the fastest for Flexible Skylines computation.

SFS (Sort Filter Skyline). The SFS algorithm is an improvement of the BNL algorithm. . The algorithm is very similar to BNL, except that there is a pre-processing of the data in which the tuples are sorted using a monotone function f. By performing this sorting, we are sure that no tuple coming after a certain tuple t dominates it. So the functioning of SFS is very simple, first the dataset is ordered using a monotone function f, after that we scan the sorted dataset and for each tuple t we check whether or not it is dominated by the tuples in SKY. If it is not dominated by any tuple in SKY then we add t to SKY.

3.2. Flexible Skyline Algorithms

3.2.1. ND computation

There are two strategies for computing ND [2], the first is to start from the skyline set and then compute the ND set (2-phase), the second is to start from the entire input dataset (1-phase). The main steps can be summarized by first choosing the number of phases: 1-phase if we want to compute ND directly from the input dataset, 2-phase if we want to compute ND after having computed the Skyline. After that, we choose whether or not to apply sorting ('U' for the unsorted version, 'S' for the sorted version) to produce a topological sort with respect to the F-dominance relation, i.e. if a tuple t precedes a tuple u in the sorted input dataset, then u $\prec_F$ t. To get the topological sort, we will use as sort function a weighted sum in which the weights are the coordinates of the centroid of the polytope W(C), where W(C) is the subset of all normalized weight vectors that satisfies C that is a set of linear constraints. Next, we choose the way in which to verify F-dominance, and to do so we have seen two alternatives: the first by solving an LP problem, and the second by verifying whether the tuple is part of the dominance region of another tuple using the vertex enumeration of the polytope W(C). We have seen that the fastest algorithm is the one-phase algorithm using the VE technique. In particular the **SVE1F** algorithm works by first sorting the dataset and then scanning it one by one. For each tuple we first check if it is simply dominated by a point in the ND set, which is initially empty, and then if not, if it is F-dominated. If it is not dominated then it is added to the ND set.

3.2.2. PO computation

For the computation of PO [2], we start from the ND set and we do the Optimality Test using the Primal PO Test or the Dual PO Test. In both algorithms, a heuristic optimization technique can be used to try to reduce the number of optimality tests, which will attempt to discard non-optimal tuples using an incremental strategy.

4. Parallel Algorithms

These algorithms works by decomposing the dataset into multiple partitions, where each partition is responsible to compute a skyline without having to consider all the points in the dataset but only its assigned points. Subsequently,

a sequential phase is performed, where the final skyline set is determined by merging the previously found local sets.

Random Partitioning. This technique has the goal of ensuring that each set Si represents a sample of S that has similar structural characteristics. To achieve this, points are randomly distributed among the various partitions, with the number of points evenly allocated to the n partitions.

Grid Partitioning. This method involves dividing the space into an nxn grid to enable parallel computation on different partitions. In this approach, each dimension is divided into n parts, resulting in a total of n^d partitions, where d denotes the total number of dimensions. Using this technique we have no control on how many points the different partitions will have, and we have limited control over the number of partitions, but manage to group the points within partitions better, managing to eliminate many points in the parallel phase compared to random partitioning.

Angle-based Space Partitioning. This technique aims to address the issue of local skylines containing an excessive number of successively dominated points and the problem of load sharing between partitions, which was observed in previous partitioning techniques. The technique involves mapping the Cartesian coordinate space to a hyperspherical space and partitioning the resulting data space into N partitions based on the angular coordinates. The angle-based approach can achieve a better balance of workload, as each partition includes both good and bad points in the data space. This results in a larger number of points being dominated in the parallel phase compared to Grid partitioning and Random Partitioning. It suffers from the same problem as Grid Partitioning where we are limited in choosing the number of partitions.

*One-dimensional Slicing (Sliced Partitioning). T*he basic idea is to overcome one of the limitations of grid and angular partitioning methods which is that we have no control over how many partitions we can use and how many points there will be in each partition. The idea of this partitioning algorithm is to sort the dataset on one dimension and divide it between the partitions equally, so that each partition will have the same number of points and we decide how many partitions to have.

5. Improvement of the Parallel Algorithms

Here we introduces techniques to improve parallel algorithms by reducing their execution time.

Representative Filtering. This technique involves selecting a few points that will act as representatives (better points) and will be used to remove as many points as possible. The one proposed in this thesis is to take the first n sorted points of each partition after the angular partitioning technique. As we will see, for independent and correlated datasets, the proposed method is better than the one proposed in [3] which, instead, selects points according to the largest dominance region, while with anticorrelated datasets, the latter succeeds in filtering more, so much that we prefer it since we will be using anticorrelated datasets for our experiments.

All Parallel Algorithm. This technique is used to avoid performing the final sequential part, by performing two parallel phases, passing in the last one to each partition the union of the local skylines found during the first parallel phase. We will see how this technique has a positive impact on the total duration of the algorithm, since the part that takes the longest is the final sequential phase in parallel algorithms.

6. Experimental Results

Our goal will be to evaluate the various parallel algorithms seen and try to understand which algorithms perform best and in what setting. The various algorithms will be run on a virtual machine with 30 cores [5]. All experiments are performed using synthetically created anticorrelated datasets. For both the skyline and ND computation (fig. 6.1 and 6.2), the best parallel algorithm is Sliced Partitioning which manages to return the smallest local set in the shortest possible time in both cases, thus impacting less on the final sequential part. Compared to the Angular Partitioning algorithm for Skyline computation, we see that in the case where an anticorrelated dataset with 4 dimensions and 3 million points is taken into account, the execution time goes from 244.97 seconds of Angular to 184.55 seconds of Sliced Partitioning. This increased speed is due to the size of the

returned local set which goes from the 66251 points of Angular to the 40251 of Sliced, thus having a lighter and consequently faster sequential phase.

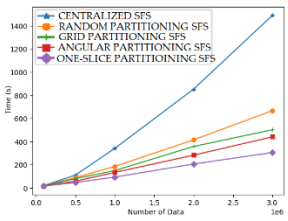

Fig. 6.1: Skyline computation

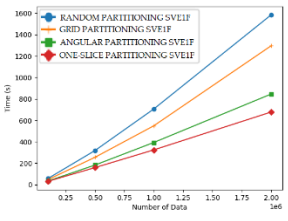

Fig. 6.2: ND computation

Representative Filtering manages to eliminate many more points bringing an advantage over the standard version by returning a smaller local set. Thanks to representative filtering, we are able to go from 184.55 seconds for the version of Sliced Partitioning without filtering to 164.1 seconds for the version with filtering in the case of the skyline computation with a 4-dimensional anticorrelated dataset with 3 million points (fig. 6.3). The best algorithm for all 3 types of skyline computation is the All Parallel Algorithm. This algorithm is the only one which, by eliminating the final sequential phase and computing the global set in parallel, manages to making the most efficient use of the parallelization and, as we shall see, will take much less time than the version of the same algorithm with the final sequential part. In this case for the skyline computation set we have a duration of 38.17 seconds, while the same algorithm with the final sequential part has a duration of 164.1 seconds which is 4.29 times longer. For the computation of the ND set instead, using a 2-million point dataset manages to finish in 99.66 seconds, about 6.5 times faster than the same version with the final sequential part (see fig. 6.3 and 6.4).

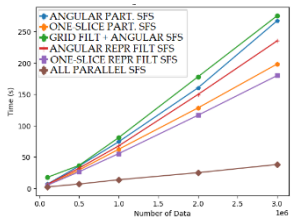

Fig. 6.3: Skyline computation using improved Algorithms

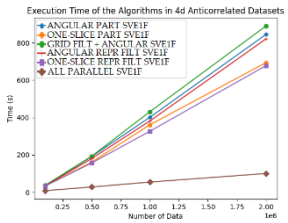

Fig. 6.4: ND computation using improved algorithms

As far as the computation of the PO set is concerned, on the other hand, the best algorithm is not Sliced Partitioning, which will perform worse than both Random and Angular Partitioning since for the computation of the PO set sorting on one dimension is not important for the computation of the global set, but Angular and Random Partitioning succeeds in dividing the load equally between the partitions (fig. 6.5). Regarding the All Parallel PO algorithm as we can see from the figure the best algorithm is Random Partitioning, which manages to finish executing the All Parallel algorithm in the shortest possible time (fig. 6.6).

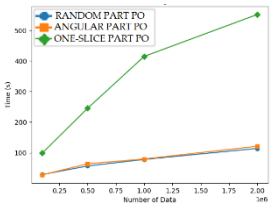

Fig. 6.5: PO parallel Algorithms

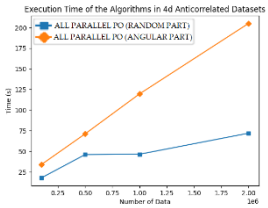

Fig. 6.6: All Parallel PO

Increasing the number of partitions in the parallel algorithms we will have a faster parallel phase but on the other hand, having more partitions the size of the local skylines returned will be greater, affecting the duration of the final sequential part. Changing the number of dimensions, instead, we have seen that the duration of the algorithms follows an exponential trend because by increasing the dimensionality of the dataset we will greatly increase the size of the global Skyline set, resulting in very high execution times. Finally, we have seen that all the algorithms, by increasing the number of cores, will decrease the execution time, with the duration of the parallel part becoming smaller and smaller as the number of cores increases, but since they always have to perform a final sequential part, this advantage is not so great. It is a different matter for PO computation, which, since it has partitions that have to compute many LPs, so we will have a parallel part with a considerable duration, thus increasing the number of cores improves the execution time by a significant amount. On the other hand, for the all parallel algorithm, which manages to make the most of parallelization, the change in the number of cores helps reduce the duration by a factor of 2.37 for the skyline computation, 2.28 for the computation of the ND set and 3.53 for the PO set.

7. Conclusions

We have introduced several parallel algorithms and applied them to the computation of Skylines and the Flexible Skyline ND and PO operators using the PySpark framework. For both skyline computation and ND, the parallel algorithms behave more or less the same in the case of anticorrelated datasets. Random Partitioning does

not perform very well because joining random points does not bring any advantage, in fact the computed local skylines is very large, thus introducing a high overhead compared to the others in the final sequential part. Grid Partitioning, on the other hand, manages to eliminate enough points in the parallel phase by returning a smaller local set than Random Partitioning, but in anticorrelated datasets, as we have seen, it has some partitions that are much heavier than others, thus incurring a fairly large computation time for the parallel phase. Angular partitioning succeeds in overcoming the problem of Grid Partitioning by being able to divide the dataset more evenly while returning small local skyline sets. Finally, Sliced partitioning succeeds in solving the problems of Grid and Angular Partitioning by managing to partition the dataset equally between partitions and still return the smallest local set of all the parallel algorithms. With the Representative Filtering the algorithms manage to return a smaller local set than the standard version, thus reducing the duration of the global set computation in the final sequential part by using a few "better" points to filter the dataset before performing parallel computation. The best algorithm however, is the All Parallel since it manages to eliminate the final sequential part, which is the longest. The algorithm manages to finish execution 4.29 times earlier than the same version with the final sequential part for the skyline computation, 6.5 times earlier for ND and 2.1 times earlier for PO.

References

[1] S. Börzsönyi, D. Kossmann and K. Stocker, "The skyline operator," Proceedings of International Conference on Data Engineering (ICDE), p. pp. 421–430, 2001.

[2] Paolo Ciaccia and Davide Martinenghi. 2020. Flexible Skylines: Dominance for Arbitrary Sets of Monotone Functions. ACM Trans. Database Syst. 45, 4, Article 18 (December 2020), 45 pages.

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

[4] Apache Spark: "Spark overview" https://spark.apache.org/docs/latest/

[5] Github repository with the implemented code: "https://github.com/emilio99-del/Master-Thesis