The Skyline Operator

A short summary by Platon Karageorgis

The following summary is discussing the paper of Stephan Borzsonyi, Donald Kossmann and Konrad Stocker which introduces the Skyline operator and claims that with certain shifts, it can be assimilated into a database system. To begin with, the authors explain the function of the Skyline operator. In a few words, this operator combines critical dimensions of a relation in order to return their strongest combinations, with respect to the purpose of the operation. This is illustrated easily with an example provided by the authors. If someone is searching for a hotel and has specific preferences like distance from the sea or low prices, the Skyline operator will return tuples that have both small distances and low prices. Considering someone knows how a database system works, it is undeniable that the Skyline operator could be of use, but so far there has not been a productive solution. Hence, the rest of the paper is dedicated to exhibit, how someone can assimilate it into a database system with good results. For a start, there is a depiction of a Skyline operator in a query, using aggregate functions like MIN, MAX or DIFF -which is supposed to state when two values should be different- and also the explanation of the semantics. In a general case, the operator should be executed after every clause except ORDER BY or any kind of clause that comes after it, which makes the integration of the operator to a query processor simpler, but exceptions of this rule are possible and investigated later in the paper. Moreover, there is a discussion about some properties of the operator, mentioning that the transitive relation is applicable, and very important to its process. In the next part, after the authors prove that a nested query cannot really replace effectively the Skyline operator, they begin to show the way it is assimilated, starting with the variant of the 2-dimensional Skyline operator. This part explains that the solution provided in the paper does not work for operators that have more than two dimensions, because they are unable to take advantage of sorting. Also, it assures that the results are satisfying when there is one or two dimensions as in the first case it is just a computation of an aggregate function, and in the second one, it is easy to take advantage of sorting and simply evaluate every tuple with the previous one. Furthermore, the algorithm of block nested loops is examined. This method uses a window in main memory, which stores tuples that cannot be compared. Subsequently, every tuple is compared to each of these window tuples. If it dominates one or more tuples, it replaces them in the window and they are discarded. If it does not dominate any one then this tuple is discarded. In case it is incomparable, it is inserted in the window list. On the other hand, if the window list is full, it is inserted in a temporary file that will be examined in the next iteration. The tuples that have been matched with others and have remained in the window at the end of each iteration can be returned,

while the ones that are in the window but have not been compared, will be potentially returned in the next iteration. The complexity of this algorithm is O(N) in the best case but unfortunately it is O(N^2) in the worst case. There are also two versions of the same algorithm depicted. The self-organizing list, that merely sorts the window list in order to accelerate the calculations and the replacing technique, where the window list maintains the stronger sets of tuples, considering them as a powerful group that will eliminate easier and faster the other tuples. In addition, there is a second algorithm discussed, divide and conquer. It divides in two equal parts the data recursively, until a partition has only one single value, and at the same time it also computes the Skyline of each part. Then, it moves on to merging all the parts, except those who are certain to be incomparable. This algorithm has the complexity of O(N*logN) + O(N*(logN)^(d-2)) which is the same in best or worse case. Some if its different versions, are m-way partitioning that divides the data into m partitions, until m is small enough to fit entirely into the memory, or the Early Skyline. The Early Skyline, fills the memory with tuples and applies the algorithm just to them, in order to discard at once the tuples that are inferior. This method increases the CPU cost but decreases the IO cost. Moreover, the authors mention shortly, the potential contribution of indices in the calculation of the Skyline operator, concluding that there is enough evidence showing that it could be a beneficial factor and therefore should be searched in the future. Additionally, there is a discussion in the case where the Skyline operator, is performed before the join operation, when the join result does not reduce the total tuples. Since the operator shares some traits with the SELECT operation and could potentially eliminate some tuples, this case should be noted. Meanwhile, it could be also beneficial within a join operation which is also explained with a paradigm. Last but not least, the authors explain the performance experiments, they mention work related with theirs and they conclude that the block nested loops algorithm is better in the best cases, but in the tougher ones divide and conquer excels.

(Note: The experimental part was skipped, as the slides state)

Which questions aren't answered by this text?

It explains simply the operations' traits and mainly the manner in which they integrated it. In addition, it includes the experimental section which proves the effectiveness of the operator. The thing that I felt was missing is the separation of the two algorithms. The paper doesn't explain when exactly we use each algorithm.

What has changed since this was written?

This text was published in 2001, so some things should be out of date in this case. I believe there should be more implementations in modern databases, and with a quick search I see the MapReduce solution, so it seems that more have been found.

Furthermore, the memory optimizations and parallel programming trend should also make a lot of this faster even though the data seem to increase too.