Pipeline Based Distributed and Anonymised Skyline Query Processing

Y. Sai Sindhu, G. Suryakala Eswari, N. Leelavathy

Dept of CSE, Asst. Professor Dept of CSE, Professor and HOD Dept of CSE
Pragathi Engineering College, Surempalem, India

Abstract

In recent ages handling of large amount of data as well as passing queries to such data is going to become challenging. More advanced query operators are essential to actionable knowledge discovery. Here actionable knowledge means the information which is used to take any decision for implementation. Skyline query processing in highly effervescent and strewn milieu requires no conformist methods or algorithms due to lack of inclusive acquaintance. This paper presents a novel approach called “Pipeline Based Distributed and Anonymised Skyline Query Processing”. Due to this strategy query processing is pipelined to speed up the process as well as version wise query results to differentiate results from time to time due to dynamic insertions or deletions in Skyline dataset. An algorithm is proposed to simulate the proposed approach.

Keywords: Skyline query, Anonymity, Pipelining, Bloom filter, R-Tree, MX-CIF Tree

Introduction

The esteem of the skyline operator is mainly due to its applicability for verdict production nature of business solutions. In such solutions, the database tuples are epitomized as a set of multidimensional points and the skyline set contain those tuples that are the best trade-offs between the different dimensions.

Consider an example of real estate database. Most of the public rank the locations in real estate by searching school, hospital, super markets, and water resources; continue power availability and transportation facilities.

Before going to enter into details of proposed approach, let us come up with some basic definitions and fruition of skyline queries.

The skyline operator was first introduced in [1], where the authors extended SQL’s SELECT statement by an optional SKYLINE OF, such that the user can specify the dimensions as well as the function (MIN, MAX, DIFF) used for the skyline query. More formally, given a data space D defined by a set of d dimensions \{d_1, ..., d_d\} and a dataset S on D with cardinality n, a point p belongs to S can be represented as p = \{p_1, ..., p_d\} where p_i is a value on dimension d_i. Without loss of generality, let us assume that the value p_i in any dimension d_i is greater or equal to zero (p_i >= 0) and that for all dimensions the minimum values are more preferable.

Skyline Definition: A point p belongs to S is said to dominate another point q belongs to S, denoted as p < q, if (1) on every dimension d_i belongs to D, p_i < q_i; and (2) on at least one dimension d_j belongs to D, p_j < q_j. The skyline is a set of points Sky(S) belongs to S which are not dominated by any other point. The points in Sky(S) are called skyline points.

Dynamic Skyline: Given a data space D and a dataset S and m dimension functions f_1, f_2, ..., f_m such that each function f_i (1 <= i <= m) takes as parameters a fraction of the coordinates of the data points, the dynamic skyline query returns the skyline set of S according to the new data space with dimensions defined by f_1, f_2, ..., f_m.

Figure 1 Distributed Skyline Processing
In paper [1] two basic in-memory algorithms has been introduced: BNL (Block Nested Loops) and D&C (Divide & Conquer). The BNL algorithm uses a block nested loop to weigh against each tuple of the database with rest of the tuples. A tuple is delivered as a result if it is not dominated by any other tuple. The D&C algorithm recursively carve up the set of input tuples into tiny sets (regions), computes the individual skyline for each region separately, and amalgamate them into the final skyline. SFS (Sort-First-Algorithm) [2] and LESS [3] improve performance of BNL by first sorting tuples according to a monotone function.

Principles of Skyline query processing:
Additive Skyline Operator Process: If every server Si is having Dataset Di locally then Sky (D) =Sky (∪Di) at query server.
Filtering: It is of two types
Data Filtering: If more than one server is having same set of points which are retrieved from neighbours it will increase process time and memory wastage. So pre-processing data points is done through pruning.
Server Filtering: All the time all the servers are not part of the queries processing due to unavailability of queried data in that peer. In that case, maintaining gateway server for filter process by comparing the query range against index of whole data. Based on the feed back some of the servers are selected for query processing.

Stages of Pipeline Based Distributed and Anonymised Skyline Query Processing

I. Pipeline Process: As described in the previous paragraphs, to increasing the processing speed and reduce the response time for vibrant data there should be a pipeline process. Initially every request is submitted to the pipe line server. Pipelining is a mechanism in which one process is dedicated to accept the request and other process is dedicated to process those requests. In the proposed approach, there are three pipeline mechanisms present. First one is take care of query acceptance and checking against cache. Second one tack care of indexing and last one take care of result aggregation and submission. Responsibilities of pipeline server includes
- Acceptance of skyline queries from clients or other servers
- Maintaining cache of recent queries and results
- Maintaining of dynamic index structure using vibrant bloom filters
- Calculate the load of each server before submitting request to backend network
- Aggregate the results retrieved from various servers

This is the actual stage added to conventional distributed sky line processing architecture. The following paragraphs include detailed explanation of those responsibilities.

1. Acceptance of sky line queries from clients or other servers: Hence number of clients from various parts of the world can send
skyline queries. Those queries will be sent to nearest servers. The data may or may not available in that server. So again those queries have to be submitted to gateway server(s) for further process. These queries are directly submitted from clients or intermediate servers.

2. Maintaining cache of recent queries and results: This will reduce the response time because if the same query is passed by any client there all the next steps will be pruned. Results are also cached, so response time is reduced. Simultaneously unnecessary re-processing and load are also reduced.

3. Maintaining of dynamic index structure using vibrant bloom filters: A Bloom filter is a space-efficient probabilistic data structure, conceived by “Burton Howard Bloom in 1970”, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not; i.e. a query returns either "possibly in set" or "definitely not in set". Vibrant bloom filter works in self-growing mode. Whenever available filter reaches its maximum size requirement, a new bloom filter is created and appended to existing list of filters. If total filters memory is greater than available data memory then it will be converted into auxiliary data structure. In general spatial skyline queries requires R-Tree index. But in the proposed approach R-Tree forest is used using these vibrant bloom filters to fit in dynamic and distributed skyline query processing.

4. Calculate the load of each server before submitting request to backend network: When ever request is accepted, gateway server has to find the load on each server. It locates the server with joint probability of load and availability of the data. This is because even though sever is free but the requested data is not in that server or it is far from the server routing and response time again increased. First priority will be given to distance and second priority to load on servers.

5. Aggregate the results retrieved from various servers: Based on the query the results may be processed by more than one server. In that case any server only contains partial results. Finally all these results from individual servers have to be aggregated at gateway server or query server.

II. Local Processing: After pre-processing is done query is submitted to related servers. Now those servers process the query individually and send the results to requested client directly or gateway server based on under laying protocol implementation.

**Algorithm to find skyline points at each server:**
**Input:** Skyline Query
**Output:** Skyline Points Dataset \((D_i)\), process time

1. Accept query from gateway server
2. Enter Radius of the skyline query point
3. Find the dominant list
4. Find the distance from query point to each dominant point
5. Filter the dominant points greater than range
6. Find the process time

**Algorithm for Vibrant Bloom Filter Building at Gateway Server**
**Input:** R-Tree Index, active-time
**Output:** Bloom Filter based R-Tree Forest

1. Build R-Tree Index at each server for spatial points along with timestamp
2. Maintain minimum and maximum values of spatial points in the R-Tree Data structure
3. Create a bloom filter with fixed size
4. Find the hash value of each R-Tree based on timestamp
5. Insert the R-Tree into current bloom filter based on hash value
6. If current bloom filter is over flow create new bloom filter and append that filter to existing bloom filter’s list.
7. If primary memory is not sufficient to handle dynamic data make older R-Tree as auxiliary R-Tree based on timestamp.
8. Now this vibrant bloom filter is treated as R-Tree forest

Simulation Results
Conclusion and Future work

Hence this paper proposes an efficient pipeline based vibrant and distributed skyline processing. Current work uses vibrant bloom filter. This is used for two purposes. Firstly, it is used as indexing structure and secondly it support offline query processing also. Skyline query processing in highly effervescent and strewn milieu requires no conformist methods or algorithms due to lack of inclusive acquaintance. This paper presents a novel approach called “Pipeline Based Distributed and Anonymised Skyline Query Processing”. According to this approach query processing is pipelined to speed up the process as well as version wise query results to differentiate results from time to time due to dynamic insertions or deletions in Skyline dataset. An algorithm is proposed to simulate the proposed approach.

Current work can be extended to process uncertain skyline query processing using pipe line and bloom filters.

References


