Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2009) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

Skyline Computation Algorithms related to Centralized Computing Environment: A Survey

R. D. Kulkarni[a], Dr. B. F. Momin[b], B.A. Ladde[c]

[a,b]Department of Computer Sceinece and Engineering,Walchand College of Engineering, Sangli, Maharashtra,India
[c]Bristlecone India Pvt. Limited,Pune,Maharshtra, India

Abstract

The purpose of this paper is to highlight the details of the various techniques involved in the computation of the skyline queries. The skyline queries have been popular in users who have multiple preferences on the various dimensions of the data. The skyline or Pareto operator selects tuples from a relation which are not dominated by the others. Although the concept of skyline computation originated from the centralized database environment, as the application needs grew, the concept has been applied successfully to modern computational environments like distributed networks, real time systems, mobile ad hoc networks etc. However, even in modern computing environments, the basic algorithms have been applied in their original or extended forms. This paper puts the focus on the various skyline computing algorithms related to the centralized computing environment and presents a summary that is useful for getting the guidelines for developing an altogether new skyline computation algorithm related to centralized or modern computing environments.

Keywords: Skyline queries, skyline computation, dominance.

1. Introduction

Prior to the introduction of the skyline queries the computational geometry had already focused on a problem known as the ‘maximum vector problem’ which aims at finding ‘maximum like hood’ of some dimension of data . The concept of skyline queries stands up on the similar line .Computing the skyline is equivalent to determine the maxima of a set of vectors. The skyline queries are popular in users who have multi preference criteria for queries. The concept of skyline operator [1] was put forward for the first time by Borzsonyi in 2001. The skyline queries can be translated into SQL queries however resultant SQL queries tend to inefficient for execution. Hence various skyline computation techniques have evolved. The skyline is defined as set of points which are not dominated by other points. This definition uses the concept of ‘dominance’. A point is said to dominate other point when it as good or better in all the dimensions and better in at least one dimensions than the other points. As an example consider a corporate office where few thousands of employees work and the employees get incentive benefits besides their salary. It will be a tedious task to find out the employees who earn minimum salary and get minimum incentives. The set of such tuples will form the skyline. Consider the sample data shown in fig. 1 below.

<employee-code, salary, incentives>

<e1, 5000, 2500>
<e2, 6000, 3000>

<e3, 6000, 1500>
<e4, 8000, 4000>

Figure 1: The sample data to demonstrate dominance

The tuple ‘e3’ dominates all other tuples on the dimension- incentives and the tuple ‘e1’ dominates all other tuples on the dimension - salary earned. However due to the multiple preferences given in the query, both the tuples ‘e1’ and ‘e3’ formulate the skyline. So for each skyline query given on the dataset, there exists a monotone scoring function ‘F’ which generates the set of skyline points ‘p’ where each ‘p’ in the skyline maximizes ‘F’. Every other point is said to be ‘out of the sky’ ! Fig. 2 demonstrates this concept.

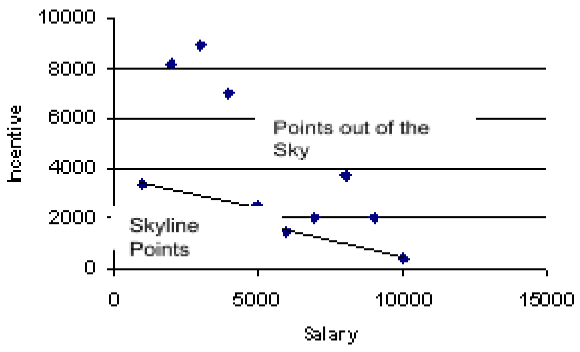

Figure 2: The Skyline of the office dataset.

The skyline query may involve two or more dimensions. For example a manager of the corporate office may be interested in finding all those employees who produce more business with minimum salary and incentives drawn. The efficiency of the evaluation of the skyline queries gets affected by the factors like dimensionality of the dataset, pre-processing of the dataset involved (if any), the environment of computing viz. centralized or distributed, the virtues of underlying network or machines viz. single or multi core, the qualities of dataset viz. co-related, independent or anti-co related, the mobility of querier and the dataset, the stagnancy of the dataset like static or dynamic etc. These and many other factors have widened the scope of research in this field. Although the modern computing environments like parallel computing, distributed computing have evolved, the study shows that new computation algorithms pertaining to these environments have their base compatible with those algorithms peculiar to centralized computing environment. Even in the recent skyline computation algorithms, basic algorithms in their original or extended versions have been applied for the new computational requirements. Hence a detailed survey of the basic skyline computation algorithms is required which will be beneficial for getting the guidelines towards the development of the new skyline computation algorithm pertaining to any of the computing environment.

In this paper, we first highlight the various related data-preprocessing techniques. We then take a review of the various algorithms related to skyline computation in the centralized environment that utilize the various features of either the data (e.g. sorted, indexed) or the computational environment (availability of cache, cores).

2. Data Pre-processing Techniques

As mentioned above the skyline queries can be translated into SQL queries however resultant SQL queries tend to be inefficient on the execution time. Hence it is always sensible to pre process the data in some way so that the skyline operator built in the query optimizer can exploit the refined qualities of the data to produce the skyline fast. These pre-processing techniques can be classified into two categories: 1) Sorting the data and 2) Indexing the data. The techniques have been discussed below.

2.1 Sorting the dataset

When computation of the skyline on raw dataset is done, the major concern in related algorithms is to reduce the possible I/O expenses. Majority of the algorithms tend to be multi pass in nature. The I/O expenses of such multi pass algorithms depend on the factors like 1) number of passes generated and 2) the size of each pass in turn. To reduce both of these factors the simplest solution employed is sorting! The process of sorting is usually applied in three ways: 1) The sorting phase: where sorting of dataset prior to beginning of the execution is done (also called as pre-sorting), 2) The filtering phase: where sorting of the qualifying subset of the tuples happens during the run of each pass and 3) The hybrid phase which is combination of sorting phase and filtering phase. These variants are discussed below.

The algorithms which make use of mere sorting phase tend to sort the dataset itself on those dimensions of the data which are compatible with the attributes of the skyline query. Then from the sorted subset generated, the skyline computation begins. As opposed to the filtering phase which aims at cutting off the non-skyline tuples from the processing, the sorting phase aims at limiting the reading of the non-qualifying tuples there by achieving the reduction in the I/O operations. If sorting is done on some monotone function f, then $f(p) \geq f(p_i)$, guarantees that the point p_i never dominates the point p. Such use of monotone function corresponds to a topological sort when related to the Pareto dominance criteria. The major advantage of this is the generation of the progressive skyline or in other words producing the incremental results without having a need of reading all the input tuples.

The algorithms based on the filtering phase (like BNL[1] and SFS[2] discussed later in section 3) make use of a small filtering window where tuples that dominate others are kept for making further dominance-test with remainder of the tuples. The earlier this filtering window can prune the dominated tuples the more it helps in getting better execution efficiency. So the window experiences sort during each pass and helps in cutting of non-skyline tuples earlier.

The sorting is not only be applied in centralized algorithms but can also be useful in distributed access techniques, where each participating system may use it to produce partial results. The final results are produced after combining partial results from all the participating systems and after some kind of processing on them. The order in which participating systems to merge the sub results can in turn prune the contribution of further participating systems and there by computing the final skyline fast. Thus the pre-processing technique of sorting is worthwhile when skyline queries are executed on top of systems which do not understand semantics of skyline queries or when the skyline computation is done on devices with limited power and/or bandwidth like mobile devices.

Let us now discuss another helpful pre processing technique ‘Indexing’.

2.2 Indexing the dataset

A database index helps in locating the required data quickly in the table. Using some key value an index structure can retrieve the required data efficiently as compared to row-by-row search method. The examples of popular index structures include B tree, B+ tree, R tree. Some of the skyline computation algorithms have also used bitmap index, BATON tree, kd tree index etc.

A skyline computation algorithm aims at utilizing the index structure to cut off non-skyline tuples from the reading and hence processing as early as possible. The more efficient the index structure is, the more efficient can be the related skyline computation algorithm. Various researchers have used index structures like BATON tree (algorithm SSP [5], algorithm iSky[8]), B tree (algorithm SUBSKY[6]) , R tree (algorithm BBS[7]) for computation of the skyline. The indexing structures can be used to index a database for an algorithm which works in centralized systems and also it can be used in distributed networks in combination like one used in centralized systems plus a separate index structure which reflects organization of the distributed network. Let us now look at some of the goals behind usage of an index structure used in skyline computation algorithms.

2.2.1 Goals of skyline computation algorithms behind using the index structure

1) Early Pruning: Generally, from the very large dataset, a small set of tuples is the skyline. So the goal behind using the index structure for the dataset is the early pruning of the non qualifying tuples. The earlier this pruning is done, the more efficient is the algorithm.
2) Accurate selection of qualifying (or possibly qualifying) tuples: When the index structure is used to index the dataset, the values of the data can be such that the overlapping regions in the index are produced. In

such cases, the skyline computation algorithms should aim at the utilization of that region (and hence the data) which can either contribute directly in the final results or would help to contribute.

The first point is self explanatory. The second point is based on the fact that many similar kinds of index structures like R tree, R+ tree, R* tree [12-13,32-33] have evolved based on the fact of the dataset features like overlapping of data. They show variance in pruning performance and coverage. (for example R* tries to reduce the overlaps using a different reinsertion method than R tree). Based on these two basic goals, many skyline computation techniques have evolved. These will be discussed in the coming section.

2.2.2 Issues to be tackled by algorithms when using the index structure

Even though the index structures offer great ease in computation of the skyline, while utilizing them following issues should be taken into account.

1) Operational cost: Using the index structure, the fast retrieval of data from the database happens at the cost of extra storage needed to store the index structure and sometimes for holding the extra copy of data. Also time required to prepare the index structure needs to be taken into account although it may be the one time task.
2) Dimensionality of tree index structures: If it is a tree index, then obvious question arises that what should be the size of each node? After a particular size of node, the performance of the index structure degrades. This 'size' factor varies for various tree structures and it is also affected by the dimensionality of the data to be indexed.
3) Conversion cost: Some times D-dimensional data needs to be transformed into N-dimensional data where N refers to node capacity of indexing structure. Although it appears to be one time task the conversion cost needs to be accounted.
4) Data Distribution: data may be horizontally or vertically distributed at various data sites in which case building the index can be a tedious task.

Irrespective of these issues, index structures are popular among the researchers as the issues can be taken care or their effect on the computational efficiency can be minimized by devising the skyline algorithm accordingly. A common observation is that, the algorithms which make use of index structures for the dataset tend to produce progressive results. There is clear difference between 'early results' and 'progressive results'. The goal behind the producing the first one is requirement of good initial response time where result generally contains tuples which are good only on few dimensions (they can be 'representative skyline'). This is not the case with the later category. Typically with large databases, satisfying user with some quick, representative results is always desired. The various criteria for behaviors and applicability of progressive skyline computation algorithms have been defined by D. Kossmann in [9]. These are:

1. Progressiveness: It should be possible to report the results to the user almost instantly they become available. The result size should increase gradually till the full skyline is computed.
2. Guarantee of zero false hits: The algorithm should not generate any points which fall out of the sky.
3. Fairness: The algorithm should be fair on all dimensions of data at the same time. In other words, no dimension should be favored during the computation of the results.
4. User control on preferences: It should be possible for user to interactively give the preferences of the dimensions involved during the computation.
5. Universality: With the same standards, the algorithm should work for all the datasets of any dimensionality.

The above features enable the researchers to choose the various computational standards while designing the technique. The next section discusses the titled survey in detail.

3. Skyline Computation Techniques in Centralized Environment

Like many other computational trends, the skyline computation originated from the centralized processing environment. The concept of skyline queries came forward in 2001. Although now the concept is widely applied to distributed environments, there have been continuous innovations in the field of software and hardware technologies formulating the centralized environment, there by keeping the skyline research open to it. This section focuses on the various aspects of computing and the related popular algorithms used to compute the skyline.

3.1 Skyline Computation Techniques: The Linear Way

By 'Linear' here, we mean to discuss those algorithms, which read and process the input dataset for skyline computation, on tuple by tuple basis or by selective reading of them. The initial techniques since the idea evolved are based either of the above strategies.

For the first time in 2001, S. Borzsonyi et. al. proposed two basic techniques namely 'Block Nested Loop' (BNL[1]) and 'Divide and Conquer' (D&C[1]) which compute the skyline by reading and processing all the input tuples. BNL maintains a *window* of incomparable tuples in main memory. When a tuple *p* is read from the input, it gets compared for dominance against all the tuples in *window* and if it is a dominated tuple it is eliminated from the consideration otherwise it may replace a tuple in *window* or being incomparable gets added into the window. If the window gets full during this process, *p* is written in a temporary file which is processed in next iterations. Obviously the BNL works well if the skyline is small. The best case time complexity of it is O (n) where n is the number of tuples in the input. And it is clear form the description that in the worst case because of nested loops used for the dominance test, the time complexity would be O (n^2). Because of this, BNL is very sensitive to dimensionality and works well for *d* up to 5 where *d* is the number of dimensions used in the query. Also due to 'replacement policy' used for in the window after the dominance test, the technique is also sensitive to correlations in the data. In summary, the BNL should be used when skyline is small, dimensionality of query is limited and dataset is anti-correlated in nature. The I/O performance of BNL depends on the number of passes made and the size of each pass. Also BNL becomes CPU bound due to costly window-comparison tests for large window size. Ad also possibility of temporary file creation can be the other issue.

The other technique D&C as clear from the name, divides the data space iteratively computing the sub-skylines for each data space until the data space is very small. Merging and elimination round happens to produce the final skyline. Because of this strategy D&C technique becomes less sensitive to the number of dimensions and correlations in the dataset. The 'Sort Filter Skyline' (SFS [2]), is also a multi pass algorithm as BNL, however the input data is sorted first topologically on the dimensions compatible with the skyline criteria, before the algorithm proceeds. BNL is greatly vulnerable to the ordering in the input however because of pre sorting in SFS, the window of incomparable tuples, does not face 'replacement' as any tuple t_{i+1} coming after t_i can not dominate it. So dominance test operations with respect to the window are less expensive than BNL. Also because only the attributes present in the skyline-criteria need only be projected during the sorting, as opposed to BNL which has to keep entire tuple, SFS only has to keep, the skyline attributes in the tuples as the algorithms proceeds. Thus input to SFS is always ordered, so SFS does not become CPU-bound with increased window size; in fact larger window size yields improved performance. An entropy function across the skyline attributes is used in sorting. Even if sort-time is accounted, SFS outperforms BNL as it executes less dominance tests.

Now it is obvious that if by some means we can further limit the tuples to be read in the window of incomparably tuples, ten it would greatly affect the efficiency of the skyline algorithms. Limiting these 'window-tuples' can be achieved by carefully choosing the sorting function and then exploiting the features of it to filter the tuples for the window. Based on this idea the algorithms 'Linear Elimination Sort of Skyline' (LESS [3]) and 'Sort and Limit Skyline algorithm' (SaLSa [4]) evolved after SFS. These two techniques also do the presorting of the input data like in SFS and also preserve all of its strengths like limiting the number of input tuples to be read, returning the results progressively, simplified management of window and optimal number of passes of the filter phase. In LESS, an 'Elimination Filter' (EF) window is applied in pass zero to eliminate the records quickly and also final pass of the sort is combined with the skyline–filter pass. In pass zero, only those tuples with the best entropy score (seen so far in the pass) are kept in EF. The dominance tests on the entropy score are executed with respect to tuples in EF and remainder of the input tuples at the first hand. The final EF then acts as a window of BNL. Because of this, size of the initial passes of LESS is smaller than the size of initial passes of SFS. Additionally a pass is saved by merging the final pass of external sort algorithms with the first skyline filter pass. Hence LESS outperforms than SFS for smaller dimensions. However the difference between them closes as the dimensionality increases as in the later case, the algorithms become CPU bound as most of the time is spent in the dominance test comparisons (LESS and SFS must make $m^2/2$ comparisons just to verify that the maximals are, indeed, maximals where m=number of maximals.). LESS has average case performance of O(kn), k being the dimension number and worst case performance of O(kn^2).Unlike SFS, SaLSa asserts, the careful choice of the sorting function. If skyline points are uniformly distributed over the data, then it is obvious that that reading those points with higher volume of skyline points increases the chance of early discarding of many other non-skyline points. So finding a 'stop point' is critical. Beyond this stop point, all the non-read input tuples are all dominated by the stop point.

In [9], an altogether different approach is used, called Bitmap. In bitmap techniques, each record is mapped to an 'm-bit vector' where m is sum of the distinct attribute values over all dimensions. For example consider the sample data with dimension d1 having 4 distinct values (4, 3, 2, 1), the dimension d2 having 3 distinct values (3, 2, 1) and the third dimension d3 having 2 distinct values (2, 1). Consider the tuple (3 2 1). The value in its first dimension is 3, which is the second largest value. So, only the bit corresponding to 4 is set to 0, while the other bits are set to 1. Because bitwise operations are fast, a series of simple operations such as and 'and, or' performed on the bit vector, reflects in bits the 'dominance strength' of a record! Due to the cost involved in mapping the data into bit vectors, the bitmap technique works well for small dimensions and smaller sets of distinct values for each dimension.

Thus all the initial algorithms evolved around the way input tuple are read, processed via some sort of ordering amongst them and they were typically used for smaller query dimensions. The success of all these algorithms depends on the ordering present amongst the data. However for all sorts of datasets and higher dimensional queries, alternate techniques were required. In the next section we focus on those techniques which utilized 'indexing' of the data and tried exploiting the benefits out of the 'structured data'.

3.2 Skyline Computation Techniques: Using the Index Structures

All the techniques described in the previous section are particularly suitable when dataset dimensions tend to be small (say d<=5). The major concern in processing when higher dimensions are involved is that of 'response time' or more keenly the 'initial response time'. It is observed that for processing those skyline queries with d>5, more efficient processing structures are needed. This is obvious because all those techniques which require reading of the whole dataset at least once tend to be inefficient when higher dimensional processing is involved. With index based techniques reading of the whole dataset is avoided as only a portion of an index is scanned to filter for the possible skyline candidate tuples. This is the beauty and the ultimate goal behind using indexing structures. The researchers' community has used various indexing structures to improve the computational efficiency. Some of the examples of the popular indexing structures used by the researchers are B tree, B+ tree, R tree[13], R* tree[12] etc. The current section highlights the techniques which use indexing for the skyline computation.

In [1], the use of B tree and R tree was suggested for the first time for the skyline computation. The B tree index was implemented in [10] for the first time. The idea put forward was to create separate index structure for each dimension specified in the skyline query. Then scanning of the index is done to get the tuples in sorted order and then filtering is done to get the final skyline tuples. However creating a separate index structure for each dimension in the skyline query is a costly operation. Also, in the techniques which use B trees, the order in which skyline is returned depends on the value distribution of the data and results may not appear 'fair'. (one or the other dimension is favored). So mapping or transformation of multidimensional data to smaller or single valued data is required (SUBSKY [6]) or a multi dimensional indexing structure like a R tree (BBS [11]) or R* tree (NN [9]) can be thought of.

Obviously the choice of using a single or multi dimensional index structure is a compromise amongst the various desired computational features of the skyline algorithms as discussed in section 2.2.2. The table in Table 1 is based on experiments carried out in [9] shows that single dimension indexing structures do well when initial response time is the concern. However the results by default favor a dimension (compromising the 'fairness') whereas the multi dimensional indexing structures do well on overall dimensions producing the 'big picture' of the results. For correlated, independent datasets single dimensional indexing structures give better response time. When the 'user control on preferences' is the concern, the multidimensional indexing structures give the better result.

Table 1: Comparison of results when 5-d, anti correlated, large database is used.

	NN	B Tree	Bitmap
Time for first 100 skyline points (secs)	8.4	0.1	23.6
Response to user interaction (secs)	0.1	~ 300	~ 87
Skyline points returned before used points found	0	15000	5117

The most simple algorithm BBS [11] used R tree to index the data and NN search to compute the skyline. In R tree index, the minimum distance 'mindist' of a point is sum of coordinates (here coordinates means values along each dimension) of the point and the mindist for minimum bounding rectangle (MBR) is the sum of its lower left corner. All points which falls into dominance region of point p are pruned directly (in other words pruning via dominance test using an enclosure query). The algorithm is efficient because the R tree fits in main memory and hence is I/O optimal. However as R tree suffers from 'curse of dimensionality' [15], BBS becomes applicable in smaller dimensions. When computation involves smaller dimensions, the algorithm outperformed all previous popular algorithms because of its applicability for all types of skyline queries as stated in [11]. Table 2 summarizes this fact.

Table 2: Applicability of algorithms to various types of skyline queries.

	D&C	BNL	SFS	Bitmap	Index	NN	BBS
Constrained	Y	Y	Y	Y	Y	Y	Y
Ranked	N	N	N	N	N	N	Y
Group by	N	Y	Y	N	N	N	Y
Dynamic	Y	Y	Y	N	N	Y	Y
K dominating	Y	Y	Y	Y	Y	Y	Y
K standby	N	Y	Y	N	N	N	Y

One more observation that can be made after discussing the data pre-processing techniques and various related algorithms is that those techniques which prune the un qualifying candidates early are obviously efficient. Most of the indexing structures are based on this principle. They try to divide the indexed data into two categories 1) possible skyline candidates and 2) un qualified candidates and try to follow the earlier category, pruning at stages the later one to get the better computational efficiency. As given in [14], this can be summarized in Table 3 below.

Table 3: Classification of basic skyline computation algorithms that use D&C and sorting techniques.

	BNL	Bitmap	D&C	SFS,LESS	Index	NN	BBS
D&C	N	N	Y	N	Y	Y	Y
Sort	N	N	N	Y	Y	Y	Y

To resolve the problem between single and multidimensional query, a possible solution can be transformation of the multidimensional tuple into a single dimensional value. SUBSKY[6] has used a simple transformation function f(p) where the transformed single dimensional value is nothing but the maximum of all distances of point 'p' from related dimensional-axis which starts at [0,1]. All the transformed single dimensional values are then indexed by a B tree. Later the algorithm ZSearch[14] was proposed which used a new index structure named ZB tree which is extension of B+ tree. The ZB tree is based on the principal of interleaving the bit-string representation of the attribute values using Z curve. The skyline computation parameters like the access order and avoidance of re-examinations of data points have direct impact on the performance of the technique. This principal matched with properties of Z curve which orders the data points and clusters them into blocks to help achieve efficient space pruning. This is because, on Z curve, dominating point is always placed before the dominated points. Obviously ZSearch outperformed BBS on I/O costs and scaled well on dimensionality for totally ordered (TO) domains (the best value for the domain is either its maximum or minimum value). The Z curve properties have been also exploited in algorithm named ZINC [16] which works for partially ordered (PO) domains (some attributes can be set-valued). ZINC (Z order Indexing with Nested Code) used a coding scheme to map the partially ordered domain into bit-strings and once done for all attributes, the usual ZB tree has been used. It differs from ZB tree only on dominance comparison tests done for PO domain attributes.

We summarize this discussion, by asserting that, the choice of index structure to be made in the skyline computation techniques is driven by the factors like dimensionality of the dataset, the type of skyline query to be applied to the dataset and the features of the dataset like totally or partially ordered dataset. Because popular indexing structure show varying performance in all above three cases (as summarized in Table 1,2 and 3 above), the researchers have to make their choice of indexing structure according to the design goal.

In the next sub section we discuss the skyline computation trends which seek the machine peculiarities to optimize the computational efficiency.

3.3 Skyline Computation Techniques: Exploiting the Machine Peculiarities

Emergence of the modern hardware technologies is continuously enabling the researchers' community to incorporate the related advances into techniques to help optimize the computational efficiency. This section discusses the skyline computation techniques that utilize the machine peculiarities such as cache and the availability of multiple cores, for getting the better computational performance.

3.3.1: Skyline Computation Techniques: Exploiting the Machine Cache

Caching of the results of the frequent user queries has always been helpful in improving the response time and I/O performance of the algorithms. This has been studied extensively in [17-20]. In concern with skyline queries, when a user fires multi preference queries, there exists a probability of repeating one or a set of attributes in sub sequent queries. For example if a user generated a skyline query asking about the charges and nearest distance from beach of the hotel, sub sequent query may ask about the distance and ratings! In such a case, if the results of previous skyline queries have been cached, they can be exploited in complete or partial query response to be given to the newly raised query by the user. Some intelligent structures like SkyCube have been proposed in [19-20] however as they exploit the correlated computational dependencies amongst the queries, their application in real time systems is difficult. Also limited cache size may not be able to work well with the SkyCube size. In this section we discuss those techniques which exploit the machine cache for improving the computational efficiency of the skyline algorithms.

The research efforts made in [23], the authors have categorized the skyline queries fired by the user as 1) an exact query: where the subsequent query is exactly same as one of the cached queries, 2) a subset query: where the subsequent query carries all the attributes of one of the cached queries, 3) a partial query: where the subsequent query carries some of the attributes of one of the cached queries, 4) a novel query: which is none of the above type. When the response to any of the skyline query is generated, a 'semantic segment' for the query is cached. For efficient search of semantics segments and avoidance of their redundant storage, an index structure 'directed acyclic graph' (DAG) has been used which is nothing but a set of attributes involved in the query, the set of skyline attributes, pointer to result table of the query, a replacement value which will be used by the cache replacement strategies, child pointers and bit vector for each attribute of the query for efficient child search. The exact and subset queries can be responded from the cache itself (with the use of the semantic segment) without accessing the database. For partial queries management of base set is done which is nothing but a set of previously cached results of a query to which new query appears to be partial. This base set already carries the partial results. The novel query is treated same as new skyline query. The cache replacement strategy makes use of the features of the semantic segment like usage frequency (age), size of skyline set and dimensionality. (In [21], the cache replacement strategy used is straightforward: LRU technique!) The experimental results have demonstrated great speed up improvements for exact and subset queries and improvement in 'initial response time' for partial queries. For the smaller datasets, the search for semantic segment is a costlier operation than directly finding the skyline from the dataset. For larger dataset, the use of technique is justified. The contribution in [26] has put forward a concept named 'user preference profile' which is cached with the query results. While generating the results for the new query a 'compatibility' function scans the cached user preference profiles and generates skyline from cached results of previous queries. The compatibility function computes the degree of closeness between the user preference profiles which are stored using DAGs. This function generates a value reflecting a match or violation of compatibility. The function helps to reduce the response time for the new query significantly by combining the cached results sets with compatible specifications. The approach outperforms the related previous work on CPU costs and I/O costs. However if the results can not be computed fully from the cached results, use of constrained skyline queries on those incompatible tuples is still required.

To summarize this discussion, we can say that a little work exists in the area of the skyline computation to exploit the usefulness of cache. The work that exists till now has utilized the interdependencies amongst the skyline results to produce the results for the new query in minimum time. The use of such techniques is needed to speed up the response time for large datasets. However, the techniques can not be applied to update intensive datasets. A research in this direction is needed. Also the trade off between size of the skyline set (larger sized results have to be removed from cache for cache is a premium asset) and dimensionality (high dimensional skyline sets in cache have high probability for serving sub sequent set) has to be tackled by right solutions.

3.3.2: Skyline Computation Techniques: Using the Multiple Cores

The advances in processor architecture have availed us with the multicore architecture where multiple independent cores share common I/O devices, the scenario, perfectly suitable for computation intensive operations. The little communication cost in parallel executing threads here, gets justified with the improved response time of the computation. In concern with the skyline computation, large number of dominance tests are required which involve just a number comparison and no I/O. Hence, the dominance tests can be processed independently. So skyline computation by its nature has a good potential to exploit the multicore architecture. So far the research efforts for parallelizing the skyline queries have been done in [24-25, 27].

The research efforts in [28] have considered the BBS algorithm as a candidate to parallelize it on multicore architecture and also have developed the PSkyline algorithm which uses the skeletal parallel programming paradigms [30] and OpenMP programming environment [31]. Algorithm PSkyline, uses divide and conquer algorithm. Given a d dimensional dataset it divides the dataset into *b* small blocks linearly. Then for each block, the skyline is computed independently (algorithm *sskyline*) and finally all the skylines are merged (algorithm *pmerge*) in parallel and processed to produce the final skyline. The parallel application of sequential *sskyline* and the sequential application of parallel *pmerge* algorithm compute the final skyline. The *pmerge* algorithm obviously performance the dominance tests while merging the two sets of skyline computed by algorithm *sskyline.* When the number of dataset blocks *b* is set to the number of cores *c*, then the whole dataset can be loaded in the main memory where optimum speed-up performance is achieved. Also parallel version of BBS is found to give results similar to *PSkyline* with c=8. However *PSkyline* outperforms on speedup with decrease in dimensionality size and the dataset size. However with respect to I/O cost and memory usage both are found to give similar performance. And also being based on brute-force principal (no index structure used for dataset and entire dataset is searched for skyline tuples) the *PSkyline* algorithm can not produce progressive results. Based on these observations, the same work has been extended in [29] where parallel versions of SFS and *sskyline* have been proposed. In parallel BBS and parallel SFS focus has been to perform the dominance tests, the most computation intensive part, in parallel. Similar to the earlier work, the results assert that when the dataset involves the most skyline tuples, the use of parallel algorithms is justified. For smaller skylines, the overhead of communication of parallel threads is considerable making the choice of these parallel versions on multiple cores less attractive. Another research work [22] has addressed in depth how the various classes of skyline computation algorithms can exploit the advantage of parallelization on multicore architecture. The authors have proposed synchronization and locking protocols (similar to that of transaction systems) for resolving the issue such as full synchronization: complete data structure is locked for every access, continuous locking: the data structure is locked at node level access, lazy locking: locks are acquired only when needed and lock free synchronization: where using a hardware instruction monitoring of any concurrent write is done and related operations are undone and repeated.. The BNL algorithm has been subjected here to locking protocols and the algorithms are *Locked* BNL, *Lazy List* BNL and *Lock Free* BNL. . The *Lazy List* parallel BNL has been implemented with *modification locks* which lock the BNL window for addition or deletion of a node. On acquiring the lock, for addition of node, in case of any concurrent modification the iteration restarts or else the node gets added to window. Another algorithm *Lock Free* parallel BNL has been implemented using the hardware instruction as explained above. When compared with *PSkyline* the parallel versions of BNL have been found performing well on main memory usage. Also *Lazy List* and *Lock Free* BNL have been found performing well on scalability.

This discussion is summarized as below:

Table 4: Summary of observations of skyline computation algorithms that use multicore architecture

	Use of multicore is recommended	Use of multicore is not recommended
1	Those skyline computation algorithms which use divide and conquer strategy for computation have high potential for parallelization on multicore architecture. They show good performance on speedup when few threads can be used. With increase in degree of parallelism, communication overheads also increase.	These algorithms can not produce progressive results. Unless all parallel operations are over and results are combined, processed and finalized, results can not be output.
2	With decrease in dataset size and dimensionality the parallelization achieved on multicore show enhanced performance. With dataset partitioned equally amongst the number of available core, the speedup advantage similar to main memory dataset are achieved.	With very large dataset, the benefits expected from parallel computations is dominated by related the communication overheads.
3	When dataset carries the most skyline tuples (most computation intensive operations), the use of parallelization gets justified. Scalability of the algorithm grows in hand with the skyline size.	When skyline size is small, the communication intensive processing dominates the benefits expected.

This summarization also includes the fact that, the algorithms like BNL, BBS, SFS have been used for testing the suitability of multicore architecture for skyline computation. However, the state of the art index based algorithms are yet to be implemented in this way.

4: Future trends:

Related to the skyline computation, there exist a number of open research issues. With emergence of new software and hardware technologies and the computational environment, the skyline computation is required in numerous numbers of applications which include categories like real time processing, mobile devices, web services etc. With very large datasets (like multimedia datasets), a proper estimate of computation cost and cardinality is expected [34-35]. A proper estimate can ease the choice of the processing methodologies. With the new laws, a new concern in this field is the 'privacy preservation'. Many online applications and mobile devices using such computations do like to preserve data privacy. So research efforts are going on in this direction [47-48]. Continuous skyline computation on update intensive datasets [43-44] (for example applications like road traffic analysis, geographic maps etc.) is another category which requires the research. The applications related to medical and stock markets also need tools which will work on uncertain or probabilistic data accurately [45-46]. Mobility of dataset and querier [40-42] need research focus in issues like adaptive, routing and range based skyline computation. In web scenario, QoS based computation, computation based on fuzzy dominance and for the SOAs are some of the new research areas [37-39].

5: Summary:

In this survey, we have put focus on the various skyline computation algorithms that work in centralized environment. We have first discussed the data processing techniques and then related algorithms. We also have discussed a category of these algorithms which use the machine peculiarities. We have tabularized the various observations made. Although, this paper has considered a limited category of algorithms suitable for centralized environments, a common observation is that, many recent algorithms in various other computing environments are still based on these surveyed algorithms. This summarizations can help and direct the development of new algorithm, in any computational environment.

References

1. 1.S. Borzsonyi, D. Kossmann, K. Stocker, The Skyline Operator, Proc. IEEE Int'l Conf. on Data Engineering (ICDE), 2001, pp. 421-430.
2. Chomicki, P. Godfrey, J. Gryz, D. Liang, Skyline with Presorting, Proc. IEEE Int'l Conf. On Data Engineering (ICDE), 2003, pp. 717-719.
3. P. Godfrey, R. Shipley, J. Gryz, Maximal Vector Computation in Large Data Sets, Proc. IEEE Int'l Conf. Very Large Data Bases (VLDB), 2005, pp. 229-240.
4. I. Bartolini, P. Ciaccia, M. Patella, SaLSa: Computing the Skyline without Scanning the Whole Sky, CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.
5. Shiyuan Wang, Beng chin Ooi, Anthony Tung, Lizhen Xu, Efficient Skyline Processing on Peer to Peer Networks, Proc. IEEE Int'l Conf. on Data Engineering (ICDE), 2007.
6. Y. Tao, X. Xiao, J. Pei, SUBSKY: Efficient computation of skylines in subspaces, Proc. IEEE Int'l Conf. Data Engineering (ICDE) 2006.
7. D. Papadias, Y. Tao, G. Fu, and B. Seeger, An Optimal and Progressive Algorithm for Skyline Queries, Proc.of ACM Special Interest Group on Management of Data (SIGMOD), 2003, pp. 467-478.
8. L. Chen, B. Cui, H. Lu, L. Xu, Q. Xu, iSky: Efficient and Progressive Skyline Computing in a Structured P2P Network, Proc. IEEE Int'l Conf. on Distributed Computing Systems (ICDCS), 2008, pp. 160-167.
9. D. Kossmann, F. Ramsak, S. Rost, Shooting Stars in the Sky: An Online Algorithm for Skyline Queries, Proc. IEEE Conf. on Very Large Data Bases (VLDB), 2002, pp. 275-286.
10. K.-L. Tan, P.-K. Eng, B.C. Ooi, Efficient Progressive Skyline Computation, Proc. IEEE Int'l Conf. Very Large Data Bases (VLDB), 2001, pp. 301-310.
11. Dimitris Papadias, Yufei Tao, Greg Fu, Bernhard Seeger, Progressive Skyline Computation in Database Systems, ACM Transactions on Database Systems, Vol. 30, No. 1, March 2005, Pages 41–82.
12. Beckmann, N., Kriegel, H., Schneider, R.,Seeger , The R*-tree: An efficient and robust access method for points and rectangles, in Proceedings of the ACM Conference on the Management of Data (SIGMOD; Atlantic City, NJ, May 23–25 1990), pp. 322–331.
13. Guttman, A. 1984. R-trees: A dynamic index structure for spatial searching. In Proceedings of the ACM Conference on the Management of Data (SIGMOD; Boston, MA, June 18–21). pp. 47–57.
14. Ken C. K. Lee, Baihua Zheng, Huajing Li,Wang-Chien Lee, Approaching the skyline in Z order, Proc. IEEE Int'l Conf. Very Large Data Bases (VLDB), 2007, Vienna, Austria.
15. K. S. Beyer, J. Goldstein, R. Ramakrishnan,U. Shaft. When Is Nearest Neighbor Meaningful? In ICDT, pages 217–235, 1999.
16. Bin Liu Chee, Yong Chan, ZINC: Efficient Indexing for Skyline Computation, Proceedings of the VLDB Endowment, Vol. 4, No. 3, 2010.

17. S. Dar, M. J. Franklin, B. T. J_onsson, D. Srivastava, M. Tan, Semantic data caching and replacement, Proc. IEEE Int'l Conf. Very Large Data Bases (VLDB), 1996, pp. 330-341.
18. Q. Ren and V. Kumar. Semantic caching and query processing. IEEE Trans. on Knowledge and Data Engineering, 15: 2003.,pp.192-210.
19. T. Xia and D. Zhang. Refreshing the sky: The compressed skycube with efficient support for frequent updates. In SIGMOD 20006, pp 491-502.
20. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube, Proc. IEEE Int'l Conf. Very Large Data Bases (VLDB), 2005, pp.241-252
21. D. Sacharidis, P. Bouros, T. Sellis. Caching dynamic skyline queries. In SSDBM 2008, pp. 455-472.
22. J. Selke, C. Lofi, W.-T. Balke, Highly scalable multiprocessing algorithms for preference-based database retrieval, in: Proceedings of the 15th International Conference on Database Systems for Advanced Applications (DASFAA), 2010, pp. 246–260.
23. Arnab Bhattacharya B. Palvali Teja Sourav Dutta, Caching Stars in the Sky: A Semantic Caching Approach to Accelerate Skyline Queries, DEXA'11 Conf. on Database and Expert systems Applications - Volume Part II.
24. S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu, Efficient skyline query processing on peer-to-peer networks, Proc. IEEE Int'l Conf. on Data Engineering (ICDE), 2007, pp. 1126–1135.
25. P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, A. E. Abbadi, Parallelizing skyline queries for scalable distribution, in EDBT, 2006, pp. 112–130.
26. Yu-Ling Hsueh Roger Zimmermann, Wei-Shinn Ku, Caching Support for Skyline Query Processing with Partially Ordered Domains, ACM SIGSPATIAL GIS ’12, November 6-9, 2012.
27. A. Cosgaya-Lozano, A. Rau-Chaplin, and N. Zeh, Parallel computation of skyline queries, in HPCS, 2007, p. 12.
28. S. Park, T. Kim, J. Park, J. Kim, H. Im, Parallel skyline computation on multicore architectures, Proc. IEEE Int'l Conf. on Data Engineering (ICDE), 2009, pp. 760–771.
29. Hyeonseung Im, Jonghyun Park, Sungwoo Park,. Parallel skyline computation on multicore architectures, Elsevier International journal of Information Systems 36 (2011), pp. 808–823.
30. F. A. Rabhi, S. Gorlatch, Eds., Patterns andskeletons for parallel and distributed computing. Springer-Verlag, 2003
31. L. Dagum, R. Menon, OpenMP: an industry-standard API for shared- memory programming, IEEE Computational Science and Engineering 5 (1) (1998) 46–55.
32. Ravi Kanth V Kothuri, Siva Ravada, Daniel Abugov, Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data, proceeding of ACM SIGMOD international conference on Management of data, 2002, pp. 546-557
33. Ynnis Theodoridis, A model for the prediction of R-tree performance,Prroc. of fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1996.
34. Surajit Chaudhuri, Nilesh Dalvi, Raghav Kaushik, Robust Cardinality and Cost Estimation for Skyline Operator, Proc. IEEE Int’l Conf. on Data Engineering (ICDE), 2007. pp. 64-66.
35. Marcel Karnstedt, Jessica M¨uller, Kai-Uwe Sattler, Cost-Aware Skyline Queries in Structured Overlays, Proc. IEEE Int’l Conf. on Data Engineering (ICDE), 2007. pp. 285-288.
36. Karim Benouaret, Djamal Benslimane, Allel Hadjali, Mahmoud Barhamg, Top-k Web Service Compositions using Fuzzy Dominance Relationship, Proc. IEEE Int’l Conf. on Services Computing, 2011. pp. 144-151.
37. Limin Pan, Liang Chen,Jian Hui,Jian Wu, QoS-based distributed service selection in large-scale web services, Proc. IEEE Int’l Conf. on Services Computing, 2011 pp. 725-726.
38. Qi Yu,Athman Bouguettaya, Efficient Service Skyline Computation for Composite Service Selection, IEEE TKDE,2011.
39. Liang Chen, Jian Wu, Shuiguang Deng, Ying Li, Service Recommendation: Similarity-based Representative Skyline, IEEE 6th World Congress on Services, 2010, pp. 360-366.
40. Hans-Peter Kriegel, Matthias Renz, Matthias Schubert, Route Skyline Queries: A Multi-Preference Path Planning Approach, Proc. IEEE Int’l Conf. on Data Engineering (ICDE), 2010, pp. 261-272.
41. Reeba Rose S,Kavitha V.R, An Efficient Location Dependent System Based On Spatial Sky Line Queries, International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Vol.1, Issue-ETIC2011, January 2012, pp. 27-29.
42. Yingyuan Xiao, Kevin Lü, Huafeng Deng, Location-dependent Skyline Query Processing in Mobile Databases, Seventh Web Information Systems and Applications Conference, 2010, pp. 3-8.
43. George Valkanas and Apostolos N. Papadopoulos, Efficient and Adaptive Distributed Skyline Computation, Proc. of Int’l Conf. on Scientific and statistical database management,, (SSDBM), 2010.
44. Yu-Ling Hsueh, Roger Zimmermann, Wei-Shinn Ku, Yifan Jin, SkyEngine: Efficient Skyline Search Engine for Continuous Skyline Computations, In Proc. Ooof ICDE, 2011.
45. Dongwon Kim, Hyeonseung Im, and Sungwoo Park, Computing Exact Skyline Probabilities for Uncertain Databases, IEEE TKDE, 2011.
46. Xiaofeng Ding, and Hai Jin, Efficient and Progressive Algorithms for Distributed Skyline Queries over Uncertain Data, in Proc. IEEE International Conference on Distributed Computing Systems, 2010, pp 149-158
47. Ilaria Bartolini, Zhenjie Zhang, and Dimitris Papadias, Collaborative Filtering with Personalized
48. Skylines, IEEE Transactions On Knowledge And Data Engineering, VOL. 23, NO. 2, FEBRUARY 2011, pp. 190-203.
49. Zhefeng Qiao , Junzhong Gu ; Lin Xin ; Jing Chen, Privacy-Preserving Skyline Queries in LBS, In Proc. Of MVHI '10 Proceedings of the 2010 International Conference on Machine Vision and Human-machine Interface, PP 499-504.