International Journal on Future Revolution in Computer Science & Communication Engineering
Volume: 4 Issue: 1
ISSN: 2454-4248
38 – 41

Query Profiler Versus Cache for Skyline Computation

R. D. Kulkarni
Department of Computer Science and Engineering
Walchand College of Engineering
Sangli, Maharashtra, India
rupaliwaje@rediffmail.com

B. F.Momin
Department of Computer Science and Engineering
Walchand College of Engineering
Sangli, Maharashtra, India
bfmomin@yahoo.com

***Abstract*—** A skyline query is multi preference user query which generates the best objects from a multi attributed dataset. Skyline computation in an optimum time becomes a real challenge when the number of user preference are large and size of the dataset is also huge. When such a big data gets queried at large, response time optimization is possible through maintenance of the metadata about the pre-executed skyline queries. We have earlier proposed, a novel structure namely 'Query Profiler' which preserves such metadata about the historical queries, raised against a dataset. Also as the dataset gets queried at large, the dimensions of user queries often overlap and queries get correlated. Such correlations in user queries and the availability of metadata about the earlier queries, combined together speed up the computation time and the optimization of the response time of the further skyline computation becomes possible. In this paper, we assert the efficacy of the Query Profiler by comparing its performance with the parallel techniques which utilize cache mechanism for optimization of the response time. We also present the experimental results which assert the efficacy of the proposed technique.

Keywords- *Skyline queries; Correlated Queries; Query Profiler*

*****

I. INTRODUCTION

The skyline queries return those objects from the dataset which dominate rest of the objects on the dimension of user's concern. The skyline queries are used extensively in the field of decision support system concerning to the areas like market research, customer information services, location based route determination etc. Skyline computation attracts the attention of the database community in the area of optimization of the response time as the datasets used for the computation are tending huge and they are also queried at large. It is also observed that, as the large number of queries are raised against the same dataset the dimensions of the user queries often overlap. For example, a person interested in buying a house may generate a skyline query for getting information about all such houses which are multistory, nearer to the schools and are cheaper. And another user may query the same dataset for finding all such houses which are nearer to the schools and are cheaper. In the above example the dimensions of the queries overlap (in fact, dimensions of the later query are subset of the dimensions of the earlier query.) Such queries are correlated queries. If the results of the earlier queries are preserved in some form, the skyline of the later query can be computed faster.

We have previously proposed a data structure namely *Query Profiler* (*QP*) [1] for managing the metadata about the historical queries raised against a dataset. In this paper, we reutilize the concept of *QP* and make new contributions as follows.

1) We compare the performance of QP with the parallel techniques which used cache memory for managing the metadata of the pre-executed skyline queries.
2) We present the experimental results and the analysis for asserting the efficiency of QP.

The remainder of the paper has been organized as follows. Section II details the literature of the parallel techniques. Section III highlights the core idea of QP in brief and Section IV elaborates the skyline computation for the correlated queries. The next Section V covers the experimental results and the related analysis and the last Section VI highlights the conclusions and the directions for the further extensions.

II. RELATED WORK

Concerning to the applicability of the skyline in the various fields, computing environments and architectures, numerous skyline computing techniques have evolved. This section briefs the major techniques in accordance with the research objective mentioned above.

The notion of skyline queries and the skyline operator [2] was presented in 2001. Since then, to suit to the era of centralized datasets, various techniques evolved for computing skyline. To speed up the computation and reduce the number of dominance tests to be carried out, few of the techniques used pre-processing techniques on the dataset like sorting the dataset, partitioning the dataset using the patterns observed in the data etc. Few techniques from this category are *Linear Elimination Sort of Skyline* (*LESS*) [3], *Sort Filter Skyline* (*SFS*) [4], *Sort and Limit Skyline* (*SaLSa*) [5]. Few techniques used efficient indexing to speed up the dataset access using various single or multi-dimensional index structures like *B* trees, *Z* trees, *R* trees, *kd*- trees etc. The techniques which used dataset indexing include *Nearest Neighbor* (*NN*) [6], *ZSearch* [7], The *Branch and Bound Skyline* (*BBS*) [8] etc. With the advancements in network architectures and the distributed storage of the dataset, the techniques opted parallel computation of the skyline. The techniques used parameters like smart distribution of the datasets at the nodes and efficient network organizations. Some of the techniques pertaining to

these features are *Distributed Skyline* (*DSL*) [9], *SKYPEER* [10], *Skyline Space Partitioning* (*SSP*) [11] etc. With the evolution of the parallel programming paradigms like MapReduce, few of the traditional skyline computing techniques were converted to suit to this paradigm as mentioned in [12]. And the other similar techniques are *MR-Angel* [13] and *SKY-MR* [14]. With the advent of modern hardware like multicore processors, *GPUs*, *FPGA*s the skyline techniques were developed to exploit the best features of these hardware. Such techniques include *GPU based Nested Loop* (*GNL*) [15], *SkyAlign* [16], *FPGA* based techniques as in [17-18], *SMS* based techniques [19].

Our contribution which uses *QP* for computation of the skylines, differs from these research efforts in various ways. First of all, unlike the caching approach mentioned in [20], our technique uses the properly indexed and managed data structure *QP* [1] to avoid the limitations of the cache control. In a busy system, it is unsafe to rely on the metadata managed in the cache as it may get wiped off. Secondly, through use of *QP* the computation technique becomes free from any sort of data re-processing and complex memory management requirements as done in [10, 21-22]. By making use of *QP*, the correlations observed in the skyline queries and the metadata of the pre-executed skyline queries is used together to optimize the response time of further skyline computation.

The basic concepts have been detailed in the next sections III and IV. Without loss of generality, the discussions made in the next sections assumes that a word ‘query’ refers to a ‘skyline query’Ease of Use

III. Query Profiler

Query Profiler (*QP*) is a main memory data structure which aims at preserving the metadata of all the queries raised against the dataset. It is managed in the main memory and it has fields like *QP = {QId, Att, S, Sb, Pr, Qf}* where *QId* is an unique query id for each of the skyline query that is executed, *Att* is a set of dimensions that occurred in the query, *S* refers to the computed skyline, Sb is a set of all those *QId*s to which the query attributes appeared as subset, *Pr* is a set of all those *QId*s to which the query attributes appeared as partial set (explained shortly in the next section) and *Qf* refers to the frequency of the occurrence of the query.

This *QP* is a well-organized data structure that has single entry for all the unique queries and it is all well-indexed using a hash index. This allows fast retrieval of the metadata of any entry. *QP* is also sorted in the background on wo parameters viz. the query frequency *Qf* and the set of query attributes *Att*. The first parameter allows the most popular queries to be on the upper side in *QP* and the later parameter keeps the scope for quickly determining the type of correlations, the query generates with the other queries.

Using the above management strategies, size of *QP* is kept to the least and also retrieval of the metadata is served fast. Both these features make *QP* an efficient structure and a quicker assistance in skyline computation of the correlated queries. The details of the same has been discussed in the next section.

IV. Skyline computation for correlated queries

When the dataset gets queried at large, chances are high that the dimensions of the queries often overlap. Such overlapping generates the correlations in the skyline queries raised by the users. These correlations are of following types.

1) An Exact correlation: It exists when a quey carries all the dimensions exactly same as that of some earlier query executed on the dataset.
2) A Subset Correlation: It exists when a quey carries few of its dimensions which are subset of dimensions of one or more earlier queries.
3) A Partial Correlation: It exists when a quey carries few of its dimensions which are subset of dimensions of some previous queries and one/more newer dimensions are present in the query.
4) A Novel Correlation: It exists when a query carries those dimensions which have never come in the dimesions of any of the earlier query.

When such correlations exist in the user queries, and metadata about the results of earlier queries is available in the form of *QP*, the computations of the skylines of the further queries is done as elaborated next.

The skyline for a query which has an exact correlation is returned from *QP* itself as it happens to be a repeat query and its skyline already exists in *QP*. The skyline for a query which has a subset correlation is nothing but the intersection of the skylines of all those queries with whom the current query has a subset type of correlation. Thus, for the queries which carry exact or subset correlations, use of *QP* makes possible the complete avoidance of the dataset access and skylines are returned immediately. As the dataset access and the costliest operation of dominance tests gets eliminated, the response time is improved. The dataset access, however cannot be avoided for the other two kinds of correlations viz. partial and novel correlations as there exists occurrence of newer query dimensions. The skyline for a query which has a partial correlation is computed as described next. The union of the skylines of all those queries with whom the current query persists a partial kind of correlation is computed first. This union becomes the first window of tuples to carry out the dominance tests amongst the dataset tuple and assists to gear up the computation. And for the skylines of the queries which have novel correlations, the dataset access is obviously mandatory.

Given this elaboration, it is clear that use of *QP* improves the response time of the skyline computation. There exist techniques like [20] which use cache memory for maintaining such metadata of the pre-executed queries. However, use of cache has following limitations.

1) The data managed in cache cannot be considered to be available at all the times and cache gets cleared and re-used to make room for newer data.
2) For a huge dataset and queries having large number attributes, more and more cache memory is required as the number of queries increase. And there exists a limitation on amount of cache allocations.

 ISSN: 2454-4248
38 – 41

3) Naïve cache segments are not indexed and hence the data retrieval time gets affected.
4) The cache control becomes even poorer when the distributed computing environment exists for the skyline computation.

As against to these points, the management of *QP* s benefited a lot from better, cheaper availability of main memory and being hash indexed the retrieval time of data is much better than the naïve cache segments. Also, the concept of *QP* has been extended by us in further research efforts detailed in [23-24] for the distributed as well as update intensive environments. This efficacy of *QP*, is implied by the experimental results. The next section covers the experimental work and the related results.

V. EXPERIMENTAL WORK AND RESULTS

The aim of these experiments is to compare the response time of the skyline computation when *QP* is used to maintain the metadata of the pre-executed skyline queries against the use of cache memory used for the same purpose. The experimental setup used for these experiments incorporates, an Intel core i-3 2100 CPU, 3.10 GHz with 2GB RAM and with the Windows 7 environment. The traditional *Block Nested Loop* (*BNL*) [2] has been used for the skyline computation. Total two experiments have been carried out. A centralized dataset available at *www.basketball.com* is assumed for the experiments. Both these experiments compare the performance of three techniques: 1) *QP*: technique which implements *QP*, 2) *JC*: technique which has implemented *JCS* cache and 3) *NQP*: technique which computes the skyline without assistance of any metadata like the one maintained in *QP*. The parameters set for these experiments are: *n*: dataset cardinality (number of tuples), *d*: dimensionality of the dataset and q: number of queries.

The first experiment compares the response times of these two techniques, when the parameter of the number of queries is varied. The parameters set for this experiment are: n=3925, d=5 and the parameter q is varied from 5 to 35. The following Fig. 1, depicts the results obtained.

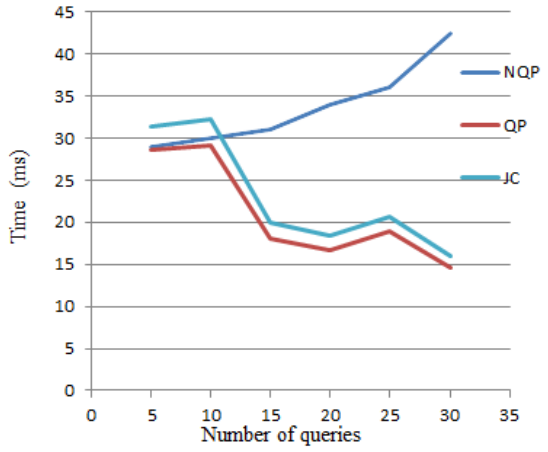

Figure 1. Effect on response time upon variace of number of queries

As shown in the Figure 1 above, the *QP* technique delivers the optimum performance. This performance is justified by the fact that, due to use of *QP*, the dataset access is cut for the queries correlated by the exact and subset correlation. Also as *QP* is well indexed, it exhibits better performance over the naïve cache segments used in the *JC* technique. And as the *NQP* technique makes the dataset access for each of the skyline computation (without taking the benefits of the correlations observed amongst the queries) it exhibits the worst performance.

The second experiment studies the same comparison when the parameter of the dataset cardinality is varied. The parameters set for this experiment are: *q*=25, *d*=5 and the parameter *n* is varied from 3000 to 21,000. The results observed have been shown below in Fig. 2.

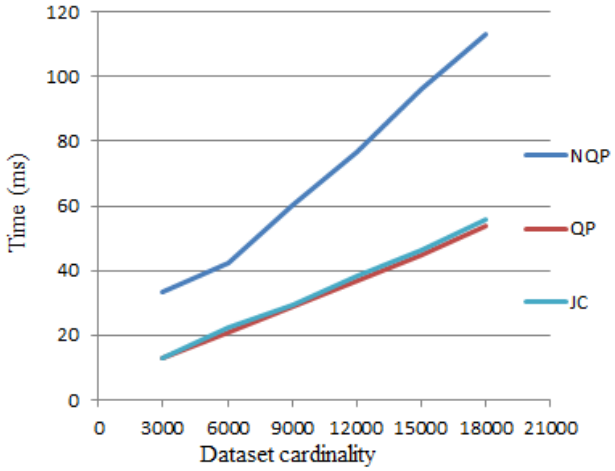

Figure 2. Effect on response time upon variance of dataset cardinality

The results shown in Figure 2 above, assert that the QP technique delivers the optimum performance. This performance is vindicated by the fact that, when the number of queries (correlations) are kept constant, the role of indexing in metadata retrieval is vital. And QP is hash indexed as against the naïve cache segments used in the JC technique. Again, the NQP technique with no assistance of metadata, shows the worst performance.

The inferences drawn from the above two experiments are analysed as next. If total *q* number of queries are executed against the dataset that contains *n* number of tuples, then the time complexity for the techniques that access the dataset for each skyline computation. Becomes *O* (*n***q*). However, when queries are correlated there is always scope for the optimization of the response time. Let *qe* and *qs* denote the number of queries that are correlated by the exact and subset correlations respectively and let *qp* and *qn* denote the number of queries that are correlated by the partial and novel correlations respectively. Then, the time complexity for the techniques that use *QP* is denoted as *O* (*n***(qp*+*qn)*) + *O* (*qe*_+*qs*). This is obvious as the dataset access avoided (in case of the exact and subset correlations) and it is done only when mandatory (in case of the partial correlations). So, we observe that, with the assistance of *QP* this becomes possible and the optimization of the response time of the skyline computation is achieved.

The next Section VI concludes the discussion.

VI CONCLUSIONS AND FURURE SCOPE

In this paper, we revisited the concept of *QP*, proposed earlier by us and compared its performance with the techniques that utilize cache memory for maintenance of the metadata of the pre-executed skyline queries. We exhibited this comparison, by means of the experiments carried out and presented the analysis of the techniques.

We find that, when a dataset gets queried a lot, the correlations exist in the user queries. These correlations prove very much useful to optimize the response time of further skyline computation when the metadata of the historical queries raised against the dataset is preserved in an efficient manner as done in *QP*. The use of cache mechanism, done for the same purpose suffers from the several limitations.

The concept of *QP* can be further be tested in parallel computing environments as well as in the scenario where the dataset experiences parallel updates. The concept can also be implemented through the parallel programming paradigms like MapReduce. In near future, we would like to extend the work on the above aspects.

REFERENCES

[1] R. D. Kulkarni and B.F. Momin,"Skyline computation for frequent queries in update intensive environment", Elsevier Journal of King Saud University - Computer and Information Sciences, Vol. 28, No. 4, pp.447-456.

[2] S. Borzsonyi, D. Kossmann and K. Stocker, "The skyline operator", Proc. IEEE Conf. on Data Engineering, 2001, pp. 421-430.

[3] P. Godfrey, R. Shipley, J. Gryz, "Maximal vector computation in Large Data Sets", Proc. Conf. on Very Large Databases, Trondheim, Norway, 2005, pp. 229-240.

[4] J. Chomicki, P. Godfrey, J. Gryz, D. Liang, "Skyline with presorting", Proc. IEEE Conf. on Data Engineering, 2003, pp. 717-719.

[5] I. Bartolini, P. Ciaccia, M. Patella, "SaLSa: Computing the skyline without scanning the whole sky", Proc. Conf. on Information and Knowledge Management, 2006, pp. 405-411.

[6] D. Kossmann, F. Ramsak, S. Rost, "Shooting stars in the sky: An online algorithm for skyline queries", Proc. Conf. on Very Large Databases, 2002, pp. 275-286.

[7] K. Lee, B. Zhang, H. Li, W.-C. Lee, "Approaching the skyline in Z order", Proc. Conf. on Very Large Data Bases, 2007, pp. 279-290.

[8] D. Papadias, Y. Tao, G. Fu, B. Seeger, "Progressive skyline computation in database systems", ACM Trans. on Database Systems, Vol. 30(1), 2005, pp. 41–82.

[9] P. Wu, C. Zhang, Y. Feng, B. Zhao, D. Agrawal, A. Abbadi, "Parallelizing skyline queries for scalable distribution", Proc. Conf. on Extending Database Technology, 2006, pp. 112-130.

[10] T. Xia, D. Zhang, "Refreshing the sky: The compressed skycube with efficient support for frequent updates", Proc. Conf. on Management of Data, 2005, pp.493-501.

[11] S. Wang, B. Ooi, A. Tung, L. Xu, "Efficient skyline query processing on peer-to-peer networks", Proc. IEEE Conf. on Data Engineering, 2007, pp. 1126-1135.

[12] B. Zhang, S. Zhou, J. Guan, "Adapting skyline computation to the MapReduce framework: Algorithms and experiments", Proc. Conf. on Database systems for Advanced Applications, 2011, pp. 403–414.

[13] L. Chen, K. Hwang, J. Wu, "MapReduce skyline query processing with a new angular partitioning approach", Proc. IEEE Conf. on Parallel and Distributed Processing Symposium, 2012, pp. 403–414.

[14] Y. Park, J.-K. Min, K. Shim, "Parallel computation of skyline and reverse skyline queries using MapReduce", Journal on VLDB Endowment , Vol. 6(14), 2013, pp. 2002–2013.

[15] W. Choi, L. Liu, B. Yu, "Multi-criteria decision making with skyline computation", Proc. IEEE Conf. on Information Reuse and Integrationpp, 2012, pp.316-323.

[16] K. Bøgh, I. Aasent, M. Maghni, "Efficient GPU-based skyline computation", Proc. Wksp. on Data Management on New Hardware, 2013, Article no.5.

[17] L. Woods, G. Alonso, J. Teubner, "Parallel computation of skyline queries", Proc. IEEE Conf. on Field-Programmable Custom Computing Machinesx, 2012, pp. 1-8.

[18] L. Woods, G. Alonso, J. Teubner, "Parallelizing data processing on FPGAs with shifter lists", ACM Trans. on Reconfigurable Technology and Systems, Vol. 8(2), 2015.

[19] R. Anurag, A. Bhattacharya, "SMS: Stable Matching Algorithm using Skylines", Proc. Conf. on Scientific and Statistical Database Management, 2016, Article No.24.

[20] A. Bhattacharya, P. Teja, S. Dutta, "Caching stars in the sky: A semantic caching approach to accelerate skyline queries", Proc. Conf. on Database and Expert Systems Applications, 2011, pp.493-501.

[21] J. Lee, S. Hwang, "QSkycube: Efficient skycube computation using point-based space partitioning", Journal on VLDB Endowment, 2010, pp.185-196.

[22] J. Pei , "Computing closed skycubes", Journal on VLDB Endowment, 2010, pp. 838-847.

[23] R. D. Kulkarni, B.F. Momin,"Parallel skyline computation for frequent queries in distributed environment", Proc. IEEE Conf. on Computational Techniques in Information and Communication Technologies, 2016, pp. 95-102.

[24] R. D. Kulkarni, B.F. Momin,"SCP: Skyline computation planner for distributed, update intensive environment", Springer SIST Series, Information and Communication Technology for Intelligent Systems, (ICTIS-2017) Vol.(1), 2017, pp.399-408.