Dynamic Optimization Algorithm of Static Materialized Views

Baili Zhang[1,2,3], Yuhang Wu[1], Linmu Wang[1], Jie Wang[1], Jianhua Lu[1,3]

[1] School of Computer Science and Engineering, Southeast University, China
[2] Key Laboratory of Computer Network and Information Integration of Ministry of Education, Southeast University, China
[3] Research Center for Judicial Big Data, China
zhangbl@seu.edu.cn, 15895923029@163.com, {657440248, 291210056}@qq.com, lujianhua@seu.edu.cn

Abstract

In this paper an algorithm DCO (dynamic cache optimization) is proposed for the static materialized views that lack better response performance to dynamic queries. Based on the optimal static materialized views, DCO combines cache mechanism to obtain dynamic materialized views with dynamic adaptability and quick response. Meanwhile, a novel method of allocating free space is presented to implement the dynamic cache, and thus the free space of system can be used efficiently to store more materialized views for improving the query response. Experimental results indicate the efficiency and feasibility of DCO, and also show that DCO can overcome the SPSE (space-performance saturation effect) of materialized views in some degree.

Keywords: Data warehouse, Materialized view, Dynamic cache

1 Introduction

Materialized view means pre-calculating and storing physically a part of the query views in the data warehouse, which can effectively speed up the response of the data warehouse to the queries. Up to now, it has become an important method to improve the multi-dimensional analysis performance of the system [1-2]. However, the selection of materialized views set, always belongs to the NP-hard problem and has always been a hot spot in the data warehousing community. For this reason, the researchers have proposed several solutions, among which, the BPUS (benefit per unit space) algorithm proposed by Harinarayan et al. is a cubic-based greedy algorithm [3]; Gupta systematically elaborates on the materialized view selection problem and proposes a series of heuristic algorithms [4]; the problem of materialized view selection is studied as space transformation problem in [5]; in [6-8], the genetic algorithms are used to obtain optimal materialized views set, and some works are conducted on reducing the complexity of the algorithms.

The above algorithms are all based on the assumption that the query distribution is predicted in advance. They essentially can be categorized into static algorithms. Due to the time varying of data warehouse, especially lots of ad-hoc queries in decision support applications, the query mode of the system becomes hard to predict, and the materialized views set selected by the static algorithm is away from its optimal solution gradually. This means the administrators need to discover the changes of query mode in time and select the appropriate time to re-execute the view selection algorithm. For a data warehouse with complex user requirements, this work is quite complex and time-consuming. Therefore, some static algorithms were modified to run online reluctantly [9-10], then the materialized views set can be automatically adjusted periodically. This reduces the workload of the administrators to some extent. However, this method requires certain statistical data before it can be performed. So the materialized views set cannot make specific adjustments for the change of the query distribution or ad-hoc queries within a certain period. Accordingly, literature [11] proposes an innovative real-time adjusting algorithm named FPUS (frequency per unit space) on the basis of improved static algorithms. However, the overhead is relatively large for the algorithm needs to compare the overall materialized benefits after each query response. It is unsuitable for the situation of high query density especially. Additionally, frequent thrashing may occur in some views, which results in the instability of materialized views set and optimized query schemes can't be utilized repeatably. Therefore the query costs are increased, and make the algorithm lose its truly practical value to some extent.

In line with the dynamic cache mechanism, DCO (dynamic cache optimization) is presented in this paper, including DCO-S algorithm based on space constraints and DCO-SM algorithm based on constraints of space cost and maintenance cost. Assisted by static algorithm that obtains the static materialized views set, DCO, as a

*Corresponding Author: Yuhang Wu; E-mail: 15895923029@163.com
DOI: 10.3966/160792642019092005012

complementary algorithm, uses dynamic cache mechanism to achieve an extra dynamic materialized views set. Then it can adapt to the changes of query distribution and ad-hoc queries, and can render the entire materialized views set with better dynamic adaptability. Meanwhile, due to the adoption of different selection mechanisms, the dynamic materialized views set, to some extent, can avoid the space-performance saturation effect (SPSE) [12] that the performance of algorithm no longer changes with the increase of materialization space. Then within a larger range, increasing the materialization space can make the materialization benefits increase continuously. In addition, DCO can adopt a more novel space allocation method to implement the cache. Referring to the mechanism of recycling bins in operating system, this method does not occupy the limited data area of the database, but uses the remaining hard-disk space of the system as the cache space. Finally, benefiting from the plug-in technology, DCO has excellent portability and compatibility and can be applied to other category of data warehouse.

2 DCO Algorithm

2.1 Problem Description

In this paper the materialized view set selected by static algorithm is referred as the static materialized set, represented by M_S. The materialized view set determined by DCO is called the dynamic materialized set, represented by M_D. The query cost definition based on the materialized view set is given as follows.

Definition 1. If the frequency of each query q_i in user query set Q is f_{q_i}, the query cost of obtaining q_i from the materialized view set M is $C_{q_i}(M)$. The query cost for the entire user query set Q is $C(Q, M) = \Sigma_{q_i \in Q} f_{q_i} * C_{q_i}(M)$.

a. If materialized view set M only includes M_S, $M = M_S$, then

$$C(Q, M) = C(Q, M_S) = \Sigma_{q_i \in Q} f_{q_i} * C_{q_i}(M) \quad (1)$$

b. If materialized view set M includes M_S and M_D, $M = M_S \cup M_D$, then

$$\begin{aligned} &C(Q, M) = C(Q, M_S \cup M_D) \\ &= \Sigma_{q_i \in M_D} f_{q_i} * C_{q_i}(M_D) + \Sigma_{q_i \in Q - M_D} f_{q_i} * C_{q_i}(M_S) \end{aligned} \quad (2)$$

Due to the contribution of M_D, a part of the query set can be responded by matching directly the materialized views in M_D instead of obtaining originally from M_S. The query cost of the entire user query set Q will bring the following changes:

$$\begin{aligned} \Delta C &= C(Q, M_S) - C(Q, M_S \cup M_D) \\ &= \Sigma_{q_i \in M_D} f_{q_i} * C_{q_i}(M_S) + \Sigma_{q_i \in Q - M_D} f_{q_i} * C_{q_i}(M_S) \\ &\quad - \Sigma_{q_i \in M_D} f_{q_i} * C_{q_i}(M_D) - \Sigma_{q_i \in Q - M_D} f_{q_i} * C_{q_i}(M_S) \\ &= \Sigma_{q_i \in M_D} f_{q_i} * (C_{q_i}(M_S), C_{q_i}(M_D)) \end{aligned} \quad (3)$$

Definition 2. If the query cost of q_i obtained from M_S is $C_{q_i}(M_S)$, its query cost turns into $C_{q_i}(M_D)$ with the help of the dynamic materialized views M_D. Because of the cache optimization mechanism, the benefits produced by materialized q_i are $C_{q_i} = C_{q_i}(M_S) - C_{q_i}(M_D)$. If the frequency of each q_i in user query set Q is f_{q_i}, then the benefit of the entire query set from the materialized set M_D is

$$\begin{aligned} &B(Q, M_S, M_D) = \Sigma_{q_i \in M_D} f_{q_i} * (C_{q_i}(M_S) - C_{q_i}(M_D)) \\ &= \Sigma_{q_i \in M_D} f_{q_i} \end{aligned} \quad (4)$$

Existing benefit models of disk cache come from memory cache, which do not consider the cost $C_{q_i}(M_D)$ of retriving q_i from disk. In contrast, the model in this paper is strict, and has taken full account of I/O costs and view retrieval costs for disk. Meanwhile, the assumptions that any query is only relevant to one materialized view in [2, 9] are not adopted in this model, so the model objectively reflects the correlation between the query cost and its view size and the entire materialized view set size.

Definition 3. Given a static materialized set M_S and user query set Q, the dynamic cache optimization is to select a materialized set M_D under certain constraints and make the benefit $B(Q, M_S, M_D)$ maximal.

Finding the maximum value of $B(Q, M_S, M_D)$ is often constrained by some conditions such as space and maintenance costs. For this reason, we will discuss them separately in the following.

2.2 DCO-S Algorithm based on Space Constraint

First, DCO-S algorithm is proposed to obtain the optimal materialized set M_D under the cache space constraint. The direct materialized view of a query is defined in order to describe the algorithm.

Definition 4. If a query has a direct corresponding view in the materialized set (including M_D and M_S), it can be obtained without cutting, slicing, drilling down and rolling up from the current views, then this view is a direct materialized view (direct_view (q_i)) of this query.

Algorithm 1. DCO-S Algorithm

Input: M_S, q_x /* q_x is a query belonging to Q */

Output: M_D

1. **if** direct_view (q_x) $\in M_D \cup M_S$ **then** return
2. **For** each $q_i \in M_D$
3. $\Phi(q_i) = fq_i * B(q_i) / s_{q_i}$ /* s_{q_i} is the size of query result */
4. Sortby $\Phi(q_i, SignA)$ /* storing id, Φ and s_{q_i} of q_i into the array SignA, and sorting by Φ*/
5. $S_D = S_D + s_{q_i}$ /* S_D is the total sizes of all views in the cache. The initial value is 0.*/
6. **end for**
7. $S_{free} = S_{cache} - S_D$ /* S_{cache} is the total size of the cache space, S_{free} is the remaining space in the current cache */
8. $S_{q_x} = sizeof(q_x)$ is the size of q_x waiting to store into the cache */
9. pre_evit={$\varnothing$} /* Pre-deleted view set */
10. count=0
11. **while** $S_{q_x} > S_{free}$ **do** /* Not enough free space to store this view */
12. count++
13. $q_j = signA[count].id$
14. pre_evit=pre_evit $\cup \{ q_j \}$ /* sorted by Φ, the view is set to be pre-deleted one by one until the free space is large enough */
15. $S_{free} = S_{free} + SignA[count].s$
16. **end while**
17. $\Phi(q_i) = f_{q_i} * B(q_x) / s_{q_x}$
18. $\Phi(pre\_evit) = \max(\{\Phi(q_i) \mid q_j \in pre_{evit}\})$ /* Get a view of the pre-deleted set with the largest Φ value */
19. **if** $\Phi(pre_{evit}) < \Phi(q_x)$ **then** /*Ensure that q_x is optimal*/
20. Del pre_evit from M_D
21. Insert q_x *into* M_D
22. **Return**.

The differences between DCO-S and the algorithm in literature [13] include not only the model, but also the method, which guarantee the optimality of the DCO-S. Under the constraints of the cache space, optimizing the materialization benefits of M_D can be expressed by the following:

$$Max(\Sigma_{q_i \in M_D} f_{q_i} * B_{q_i}(M)) \quad \textbf{(5)}$$

$$\Sigma_{q_i \in M_D} s_{q_i} \le S_{cache} \quad \textbf{(6)}$$

The problem defined by formula (5) and formula (6) belongs to NP-complete backpack problem. There is no effective algorithm for this type of problem, but if the size of the cache s_{cache} is large enough compared to the size of a single view s_{q_i}, s_{cache} can almost be filled up by the small s_{q_i}. In this way, formula (6) can be modified to

$$\Sigma_{q_i \in M_D} s_{q_i} = S_{cache} \quad \textbf{(7)}$$

For the problems defined by equations (5) and (7), it can be proved that the algorithm DCO-S can obtain the optimal solution.

Lemma 1. The materialized set obtained by the DCO-S is the optimal solution to the problem defined by formula (5) and (7).

Proof. Omitted for the length limitation of paper.

For the calculation of the real-time frequency f_{q_i}, the literatures [13-14] learn from the LRU-K mechanism [15]. The calculation considers the latest K references to this query (generally 1≤K≤5), avoiding the shortcomings of using a single reference for recent estimates. This is a relatively good way to express instant frequency, but it ignores the long-term distribution of queries. In literature [14] the sum of query times is added to consider the long-term cumulative value. However, if the queries do not occur during the accumulation period, the cumulative value is cleared to zero. This processing method appears a bit rough and can cause frequency change sharply. For this reason, this paper improves the method and proposes an algorithm AIF (attenuator of instantaneous frequency), which uses the attenuation factor to gradually reduce the cumulative value to zero. That is, if the query does not occur during a shorter period,

then the cumulative number of its query is decremented by a certain attenuation coefficient. Accordingly frequency values is avoided to change sharply that cause large thashing of the materialized set. The algorithm is as follows:

Algorithm 2. AIF Algorithm

Input: q_x

Output: f_{q_x}

1. **while** any query q_x *occur* **do**
2. query_count_sum++ /* The sum of query times */
3. $q_x \cdot$ ref_count++ /* Accumulate the query times of q_x */
4. $q_x \cdot ref\_flag$ = USED
5. q_x .ts [ref_count%K]=query_count_sum
   /* Time stamp: record the time when q_j occurs. Here, the total number of the query occurrence is used as time stamp.*/
   /* Assume that the array starts at 1, K represents the number of recent references */
6. **if** $q_x, ref\_count >$ K **then**
7. $f_{q_x} = ref\_count$ * K /(query_count_sum) - $q_x \cdot$ ts[(ref_count+K)%K])
8. **else**
9. f_{q_x} =ref_count/(query_count_sum- q_x ,ts [1])
10. **If** query_count_sum%Time=0 **then** /* Time is a shorter statistical period */
11. **For** all $q_j \in Q$ **do**
12. **If** $q_j \cdot ref\_flag$ = UNUESD **then** /* q_j does not occur in this period */
13. **If** $q_j \cdot ref\_count >$ K **then** /*ξ is an attenuation factor*/
14. q_j .ref_count= q_j .ref_count%K+K*(q_j .ref_count/K/ξ);
15. **else** q_j ref_count=0
16. q_j .ref_flag=UNUESD
17. **end while**
18. **Return**.

In the end of this section, the cache replacement measurement will be discussed.

$$\Phi(q_i) = f_{q_i} * \frac{B(q_i)}{s_{x_1}} = f_{q_i} * (C_{q_i}(M_S) - C_{q_i}(M_D)) / s_{x_1}.$$

In fact, the implementation of the entire algorithm is mainly based on the calculation and comparison of the measurement $\Phi(q_i)$. This is the common feature of all cache algorithms. The difference lies in the definition of specific measurement. Compared to WATCHMAN [13], Dynamat [16], and other algorithms that are simply migrated from the memory mechanism and ignore the I/O cost $C_{q_i}(M_D)$ of reading the materialized views from the hard disk, DCO's model is strict and it doesn't ignore the I/O cost $C_{q_i}(M_D)$. Because $C_{q_i}(M_D)$ may not be much different from $C_{q_i}(M_S)$, it cannot be ignored. Otherwise, it will cause a large deviation, which will affect the choice of the optimal materializated view set.

2.3 DCO-SM Algorithm

In practice, materialized view selection is often constrained by the size of space and the cost of view maintenance simultaneously. View selection under multiple constraints is a complex issue. For this reason, this problem is simplified in light of the algorithm presented in this paper. It is classified in the following situations: (1) The maintenance window of the materialized view is large, and both M_s and M_D can be synchronously updated and maintained. The maintenance cost does not become constraints, only the space constraints are taken into account; (2) The view maintenance window is medium. When M_s is maintained, M_D can be partially maintained. In this case, the problem is transformed into a dynamic materialized view selection problem under the maintenance cost constraints. Similar studies can be carried out referring to space constraints; (3) The view maintenance window is small, only M_s can be maintained. For situation (3) is universal and practical, this paper discusses it in detail and proposes a DCO-SM algorithm that takes multiple constraints into

consideration. The algorithm is based on a strategy: synchronous updates are not adopted for M_D. Once views in M_D are required to be updated, they are simply eliminated from the cache. This approach makes the algorithm simple and practical.

Algorithm 3. DCO-SM algorithm

Input: M_S, q_x

Output: M_D

1. **if** direct_view (q_x) $\in M_D \cup M_S$ **then** return
2. **For** each $q_i \in M_D$
3. $\Phi(q_i) = f_{q_i}(1-f_{u_i}) * B(q_i)/s_{q_i}$ /*Adopt the measurement different from that of algorithm 1*/
4. Sortby $\Phi(q_i, SignA)$ /*In non-decreasing order of Φ */
5. $S_D = S_D + S_{q_i}$
6. **end for**
7. $S_{free} = S_{cache} - S_D$
8. $s_{q_x} = sizeof(q_x)$
9. pre_evit=$\{\varnothing\}$
10. count=0
11. **while** $S_{q_x} > S_{free}$ **do**
12. count++
13. $q_j = signA[count].id$
14. pre_evit=pre_evit$\cup\{S_{q_x} > S_{free}\}$
15. $S_{free} = S_{free} + SignA[count].s$
16. **end while**
17. $\Phi(q_x) = f_{q_x} * (1-f_{u_x}) * B_{q_x}/s_{q_x}$ /*The update of the measurement of q_x for eliminate impact is considered when calculating*/
18. $\Phi(pre\_evit) = \max(\{\Phi(q_j) \mid q_j \in pre\_evit\})$ /*Get the view with the max value of Φ in pre-deleted set*/
19. **if** Φ(pre_evit)< $\Phi(q_x)$ **then**
20. Del pre_evit from M_D
21. Insert q_x into M_D
22. **Return**.

There is no essential difference between Algorithm 3 and Algorithm 1. However, Algorithm 3 considers the benefits' decrease of dynamic materialized views due to updating the materialized view. the changed benefit model is $\Phi(q_i) = f_{q_x} * (1-f_{u_i}) * B(q_i)/s_{q_i}$. That is, an $(1-f_{u_i})$ item is added and f_{u_i} is the update frequency of query q_i. The measurement in Algorithm 1 is $\Phi(q_i) = f_{q_i} * B(q_i)/s_{x_i}$. Its premise is not to consider the materialized view update or to assume that the system has strong update and maintenance capabilities. When update cost of the materialized view is large and must be considered, it will lead to a large deviation in the calculation of the measurement.

For the optimization of the algorithm, it can be similarly expressed by the following optimal solution:

$$Max(\Sigma_{q_i \in M_D} f_{q_i} * (1-f_{u_i}) * B(q_i)) \quad \textbf{(8)}$$

$$\Sigma_{q_i \in M_D} s_{q_i} = S_{cache} \quad \textbf{(9)}$$

According to the problems defined by formula (8) and formula (9), we can prove that the DCO-SM can obtain the optimal solution. The lemma is as follows:

Lemma 2. The materialized set obtained by DCO-SM algorithm is the optimal solution to the problem defined by formula (8) and formula (9).

Proof. Omitted for the length limitation of paper.

2.4 Implementation of Dynamic Cache Mechanism

In order to implement the cache mechanism better, DCO proposes a more novel approach to space allocation, which learns from the implementation mechanism of the recycle bin in the operating system. Fully using the system's remaining hard disk space as the cache space can avoid occupying the limited space of the database, and also avoid conflicting with other

data in the database.This plug-in implementation technology can form a loose combination between the data warehouse and the dynamic cache. So that it will not limited to a specific data warehouse, and has a high portability and compatibility.

When the system's remaining space is limited, DCO must periodically detect the available space of the system, or intercept the alarm of other applications lacking space. Then it can dynamically adjust the available free space so that the free space of the system can be fully utilized and meanwhile other applications of the system are not interfered. When the available space is reduced and some of the views need to be eliminated, DCO can re-select M_D under the new constraint space; or the simple elimination algorithm can be used to eliminate some views according to Φ of the views until space requirements are satisfied.. The specific elimination algorithm is as follows:

Algorithm 4. Simple elimination algorithm

1. $\Delta S = S_{cache} - S'_{cache}$ /* S'_{cache} is the size of cache space after shrinking */
2. $S_{evit} = 0$
3. **while** $S_{evit} < \Delta S$ **do**
4. count++
5. $q_j = SignA[count].id$
6. pre_evit=pre_evit$\cup\{q_j\}$
7. $S_{evit} = S_{evit}$ +SignA [count].s
8. **end while**
9. Del pre_evit from M_D
10. **Return**

On the contrary, if the remaining space of the system becomes larger, the dynamic cache space can be expanded in an appropriate manner, so that more views can be materialized and the query response performance of the system can be improved.

3 Experimental Design

3.1 Experimental Design

In order to verify the effectiveness of the dynamic cache optimization algorithm, a series of performance comparisons were made in the experiments between DCO and static algorithms. As a selection algorithm of static materialized views set, PBS (pick by size) [12] is currently the fastest and most widely used one. Despite the advantage of selecting the optimal materialized views set, the latest static algorithms, such as the genetic algorithm [7-8], have the disadvantage of the high time complexity. Therefore, the static materialized views set M_s in the experiments was selected by PBS, and the dynamic materialized views set M_D was dynamically selected by DCO-S (update problem was not considered). That meant the actual comparison was between DCO-S+PBS and PBS.

The experimental hardware platform was Dell PowerEdge 6600 (Dual Xeon CPU 1.5G, RAM 4G), and the database platform was Oracle 8. The experimental data set was a data warehouse corresponding to a pharmaceutical sale company transaction database [17]. There were 4 dimension tables and 1 fact table. The size of fact table records was 1.6M. The K value of the AIF used for frequency estimation was 3, the attenuation factor was 2. The average response time of the queries was used to measure the system response performance of different materialized views set to the queries. In the experiments, the system's remaining space was 53G, which made the actual space constraints nonexistent. In order to test the effectiveness of the algorithm, two methods of changing the size of the cache space were adopted in this experiment.

3.2 Experimental Steps and Results

The experimental steps are as follows: First, the space constraint was set to 100M. And the static materialized views set M_s was determined by using the PBS. Then the simulated query generator generated 5,000 random queries using 6 query templates, and the average query response time were calculated. Second, DCO-S algorithm was invoked, the constraint space was set to 20M, and the average response time to different template query sets under M_s and M_D were calculated. The results are shown in Figure 1, where T1 represents that template 1 was used to generated queries

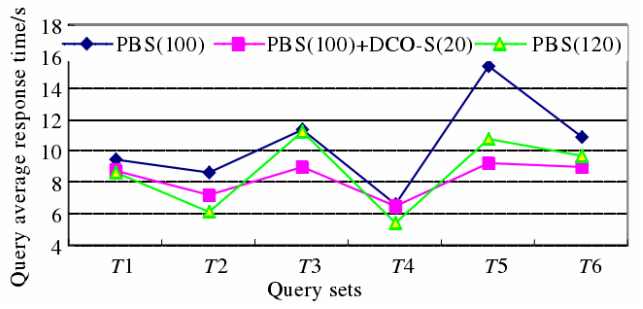


Figure 1. Comparsion of query response time (1)

Figure 1 shows that due to the help of M_D obtained by DCO-S, the query response performance of the system has been greatly improved. At the same time, the response performance to the queries generated by different templates appears to be relatively balanced. On the contrary, when the materialized views are only consisted of M_s (the static materialized views), the response performance appears great volatile. This is because the dynamic adjustment strategy of DCO-S enables the entire materialized views set to have a good adaptability. However, these changes or improvements cannot exclude the help of the increased cache space (even if this increased space utilizes the remaining space of the system). For this reason, the following experiment was conducted: M_D was deleted, DCO-S was suspended, and the space constraint was offset to 120M; M_s was reselected by the PBS, then its response time to each query set was re-calculated. The experimental results are also expressed in Figure 1. From the results, the performance improvement does not have too much difference from the PBS+DCO-S. That is said, when the total available space is 120M, there is slight difference in query performance between PBS+DCO-S and PBS, except for PBS+DCO-S having the greater advantage in balancing the different queries and tracking the ad-hoc queries.

However, when the space constraint was changed to 300M, the cache space remained to be 20M and the above three groups of experimental steps were repeated in the same order. The results are shown in Figure 2. It can be seen that PBS (300)+DCO-S (20) has a considerable increase compared to PBS (320) or PBS (300); while PBS (320) has few changes compared to PBS (300). By analyzing the space-performance curve of a static materialized views, it is not difficult to explain: When the space is small, the slope of the curve is steeper, and the space change can cause a large change in performance (by contrast, the advantages of DCO are not obvious); when the space is large enough, the curve is in the saturation region where the increase of static materialization space can not lead to the improvement of query performance. At this time, DCO appears the great advantage because it is not affected by the space-performance saturation effect in a certain range.

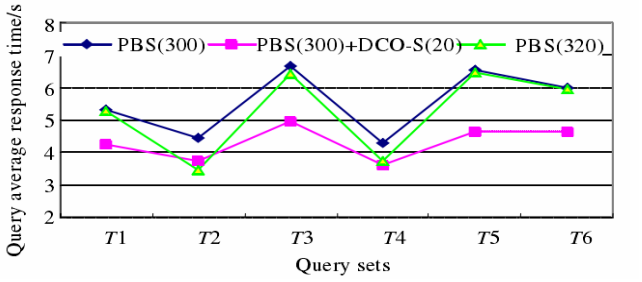


Figure 2. Comparsion of query response time (2)

It can be seen that the materialized views set optimized by DCO has a good dynamic adaptability to different query sets, and that whether the static materialized view reaches saturation or not, it can still improve the query performance. When the static materialized view is in the saturation region, the performance advantage of DCO can be better reflected compared to the same space increase.

In addition, although FPUS and PMVS can also improve the dynamic performance of the materialized views set, practically it is very difficult for them to adjust the materialized views set online Different from the real-time online adjustment performed by DCO, PMVS can only lead to the periodic dynamic adjustment of the static materialized views set. While FPUS can theoretically adjust materialized views set online, it must traverse the entire view after each query occurs. which results in a large overhead and is infeasible for large data sets.

4 Conclusion

As data warehouse is time-varying, the query response performance of the static materialized view set decreases, especially for ad-hoc queries. Therefore, this paper proposes a dynamic cache optimization algorithm as a supplementary of the static materialized view selection algorithm. The algorithm can provide effective support for dynamic queries and ad-hoc queries. At the same time, experiments have revealed that due to the combination of the cache mechanism, this algorithm can effectively overcome SPSE of the static materialized view set to some extent, making it possible to improve the system's query response performance by increasing the materialization space. By changing the size of the cache, the subsequent experiments discovered that the dynamic materialized views set also had a saturation effect. How to solve this problem would be the next work.

Acknowledgements

This work was partly supported by the National Key R&D Program of China (2018YFC0830200) , the Fundamental Research Funds for the Central Universities (2242018S30021 and 2242017S30023)

and Open Research Fund from Key Laboratory of Computer Network and Information Integration In Southeast University, Ministry of Education, China.

References

[1] A. Dhankar, V. Singh, A Scalable Query Materialization Algorithm for Interactive Data Exploration, *2016 Fourth International Conference on Parallel Distributed and Grid Computing (PDGC),* Waknaghat, India, 2016, pp. 128-133.

[2] B. Arun, T. V. V. Kumar, Materialized View Selection Using Bumble Bee Mating Optimization, *Int. J. Decision Support Syst. Technol*., Vol. 9, No. 3, pp. 1-27, July, 2017.

[3] V. Harinarayan, A. Rajaraman, J. D. Ullman, Implementing Data Cubes Efficiently, *Proc. of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, Canada, 1996, pp. 205-216.

[4] H. Gupta, Selection of Views to Materialize in a Data Warehouse, *Proc. of the 6th ICDT*, Greece, 1997, pp. 98-112.

[5] D. Theodoratos, T. Sellis, Designing Data Warehouse, *Data & Knowledge Engineering*, Vol. 31, No. 3, pp. 279-301, November, 1999.

[6] M. K. Sohrabi, V. Ghods, Materialized View Selection for a Data Warehouse Using Frequent Itemset Mining, *Journal of Computers*, Vol. 11, No. 2, pp. 140-148, March, 2016.

[7] C. Zhang, X. Yao, J. Yang, An Evolutionary Approach to Materialized Views Selection in A Data Warehouse Environment, *IEEE Trans. on Systems, Man and Cybernetics, Part C,* Vol. 31, No. 3, pp. 282-294, August, 2001.

[8] J. T. Horng, Y. J. Chang, B. J. Liu, Applying Evolutionary Algorithms to Materialized View Selection in a Data Warehouse, *Soft Computing,* Vol. 7, No. 8, pp. 574-581, July, 2003.

[9] D. Theodoratos, T. Sellis, Dynamic Data Warehouse Design, *Proc. of the 1st Int'l Conf. on Data Warehousing and Knowledge Discovery*, London, UK, 1999, pp. 1-10.

[10] B. L. Zhang, Z. H. Sun, X. Sun, Preprocessor of Materialized Views Selection, *Journal of Computer Research and Development*, Vol. 41, No. 10, pp. 1645-1651,October, 2004.

[11] H. X. Tan, L. X. Zhou, Dynamic Selection of Materialized Views of Multi-Dimensional Data, *Journal of Software*, Vol. 13, No. 6, pp. 1090-1096, June, 2002. http://www.jos.org.cn/1000-9825/13/1090.htm.

[12] A. Shukla, P. Deshpande, J. F. Naughton, Materialized View Selection for Multidimensional Datasets, *Proc. of the 24th Int'l Conf. on VLDB*, San Francisco, USA, 1998, pp. 488-499.

[13] P. Scheuermann, J. Shim, R. Vingralek, Watchman: A Data Warehouse Intelligent Cache Manager, *Proc. of the 22nd Int'l Conf. on VLDB*, San Francisco, USA, 1996, pp. 51-62.

[14] E. Otoo, F. Olken, A. Shoshani, Disk Cache Replacement Algorithm for Storage Resource Managers in Data Grids, *Proc. of the IEEE/ACM SC 2002 Conf. on Supercomputing*, Los Alamitos, USA, 2002, pp. 1-15.

[15] E. J. O'Neil, P. E. O'Neil, G. Weikum, The LRU-K Pages Replacement Algorithm for Database Disk Buffering, *Proc. of the 1993 ACM SIGMOD Int'l Conf. on Management of Data*, New York, USA, 1993, pp. 297-306.

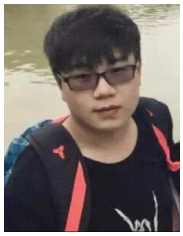
[16] Y. Kotidis, N. Roussopoulos, Dyna Mat: A Dynamic View Management System for Data Warehouses, *Proc. of the 1999 ACM SIGMOD Int'l Conf on Management of Data*, New York, USA, 1999, pp. 371-382.

[17] R. Kimball, *The Data Warehouse Toolkit,* 2nd ed., John Wiley & Son, 2002.

Biographies

Baili Zhang is an associate professor in School of Computer Science and Engineering, Southeast University, China. Before joining university, he worked as an engineer in NARI, a famous electric power research institution in China. He obtained his PhD in Computer Applications from School of Computer Science and Engineering, Southeast University. His current research focuses on (1) big data and service computing; (2) uncertain data managemen; (3) materialized view in data warehouse, and application of wireless sensor network.

Yuhang Wu received his BE degree in Nanjing Audit University, China 2017. He will continue to study for a Master's degree at Southeast University. His current research interests include interests include Natural Language Processing and deep learning.



Linmu Wang is a postgraduate student in College of Software Engineering, Southeast University, China. His current research focuses on (1) data science and data engineering; (2) natural language processing.

Jie Wang is a postgraduate student in College of Software Engineering, Southeast University, China. His current research focuses on (1) data science and data engineering; (2) natural language processing.

Jianhua Lu is an associate professor in School of Computer Science and Engineering, Southeast University, China. He obtained his Ph.D. in Computer Science from School of Information Science and Engineering, Northeastern University of China. His current research focuses on (1) data science and data engineering; (2) healthcare data analytics; (3) anomaly detection and fault diagnosis.