SLEMAS: An Approach for Selecting Materialized Views Under Query Scheduling Constraints

Ahcene Boukorca
LIAS/ISAE-ENSMA - Poitiers University
Futuroscope, 86960, France
France
boukorca@ensma.fr

Ladjel Bellatreche
LIAS/ISAE-ENSMA - Poitiers University
Futuroscope, 86960
France
bellatreche@ensma.fr

Alfredo Cuzzocrea
ICAR-CNR & University Calabria
Rende (CS)
Italy
cuzzocrea@si.deis.unical.it

ABSTRACT

Materialized views are one of the most popular optimization techniques selected during the physical phase to speed up query processing in traditional and advanced databases. Their selection has been proven to be NP-hard. As a consequence large panoply of heuristics has been proposed to find near optimal solutions. Usually, the selected materialized views are whole life disk resident and their presence is not calling into question. Note that view maintenance can cause significant amounts of CPU and I/O usage, which can be detrimental to performance in a write-intensive database application. Typically materialized views are stored on disk; however with big number of queries, there are situations where not all a good candidate views will be selected. As a consequence, their dynamic selection becomes a necessity. In this paper, we address the problem of materialized view selection by considering the query scheduling. We first review the most important existing work on static and dynamic view selection. A formalization of the problem of view selection considering the re-ordering of a large number of queries is given. A system, called SLEMAS, playing the role of a generic advisor is described. Finally, intensive experiments are conducted to compare the efficiency of our system regarding the most important state of art algorithms.

1. INTRODUCTION

Nowadays, the complex OLAP queries involving joins and aggregations is part of the scenery of applications requiring extremely large databases such as data warehouses, scientific and statistical databases. Materialized views ($\mathcal{MV}$) are one of the most popular optimization structures used in several databases deployment platforms: centralized [10], distributed [4], Cloud [12]. $\mathcal{MV}$ are used to pre-compute and store aggregated data such as sum of extremely large tables such as the fact table of a given data warehouse schema. So, $\mathcal{MV}$ are suitable for queries with expensive joins or aggregations. Once selected, all queries will be rewritten using $\mathcal{MV}$[1] in order to avoid irrelevant base table accesses. A rewriting of a query q_i of a given workload using views is a query expression q_i' referencing to these views. The query rewriting is done *transparently* by query optimizer. To generate a best rewriting for a given query, a cost-based selection method is used [2].

Note that $\mathcal{MV}$ store data from base tables. In order to keep the $\mathcal{MV}$ in the database repository up to date, it is necessary to maintain them in response to the changes at the base tables. This process of updating views is called *view maintenance* which has evoked great interest in the past years [8]. A $\mathcal{MV}$ can be either recomputed from scratch, or incrementally maintained by propagating the base data changes onto that view. Note that re-computing the views can be prohibitively expensive.

Due to resources required for $\mathcal{MV}$ (disk space, computation time, maintenance overhead and cost required for query rewriting process), DBA cannot materialize all possible views [11]. Hence, she needs to select an appropriate set of views to materialize under some resource constraints. Historically, the Views Selection Problem ($\mathcal{VSP}$) has been formalized as follows [10]: given a set of most frequently used queries $\mathcal{Q} = \{q_1, q_2, ..., q_n\}$, where each query q_i has an access frequency f_i ($1 \leq i \leq n$) and a set of resource constraints $\mathcal{CS} = \{C_1, ..., C_k\}$. The $\mathcal{VSP}$ consists in selecting a set of $\mathcal{MV}$ that minimizes one or more objectives, possibly subject to one or more constraints. Many variants of this problem have been studied that concern the studied objective functions and the resource constraint(s). The main popular variations use respectively the following objective function and constraints: (i) *minimizing the query processing cost subject to a storage size constraint* [10], (ii) *minimizing query and maintenance costs subject to storage space constraint* [15], (iii) *minimizing query cost under a maintenance constraint* [10]. This problem is known as an NP-hard problem [10]. Several algorithms were proposed to deal with this problem. In [17], a classification of existing algorithms is given. Note that commercial and academic DBMS propose tools (advisors) (e.g., DB2 design Advisor, SQL access Advisor for SQL, Data Tuning advisor for SQL server, PARINDA for Postgres [16]) to recommend $\mathcal{MV}$ to DBA.

The most traditional formalization selects views in a static where it assumes that the queries is *a priori known and pre-*

The 20th International Conference on Management of Data (COMAD), 17th-19th Dec 2014 at Hyderabad, India.

[1]This process is known as *query rewriting*

ordered. To relax this hypothesis, the dynamic selection has been proposed [14]. Existing algorithms for dynamic selection of $\mathcal{MV}$ are divided into two categories based on their incoming workload [6]: algorithms with a predefined workload (the algorithm described in [21] belongs to this category) and algorithms with an unknown workload (DynaMat system is an example of this category [14]). Typically selected $\mathcal{MV}$ are stored on disk permanently with updating strategy on these $\mathcal{MV}$; however with disk space limit and with big number of queries to be optimized, there are situations where not all a good candidate views will be selected. As a consequence, a flexible selection of the views under space constraint is recommended. It is characterized by temporary presence of the views on the disk to maximize the benefit of using all best views . In this mode of views selection, the views are created as a workload execute. If a query need a view that is not created, the view will be materialized on demand. If there is not enough space, existing views may be dropped following LRU rules. The views selection manner called dynamic materialization. To maximize the benefit of using materialized views before their dropping, it is important to permit workload's query order. this query order permutation called query scheduling.

Most important studies related to the dynamic selection ignore the query scheduling, except the Phan et al.'s [21] and Diwan et al.'s [7] works. This ignorance may penalize the performance of the selected $\mathcal{MV}$. Note that the query scheduling defines an efficient order to evaluate a set of queries to take benefit from current content of a storage device before relevant data is evicted. The device may be a main memory buffer, a secondary memory device such as hard disk, flash, etc. A couple of existing studies showed the impact of query scheduling on managing buffer where the allocation database objects into the buffer is guided by the order of the queries [9]. Recent research efforts study the impact of query scheduling on selecting optimization structures such as $\mathcal{MV}$, indexes [21] and horizontal data partitioning schema [1]. To the best of our knowledge the work of Phan et al. [21] is the sole that dynamically selects $\mathcal{MV}$ by considering query re-ordering. Consequently, we propose to details the architecture of the proposed system. It is mainly composed of four components: Dynamic Materialized Query Table (MQT) Scheduler (DMS), MQT candidate generator, scheduled queries generator, and Dynamic MQT management module, which will be described below:

- The DMS receives query workload.
- MQT candidate generator: the MDS sends the queries to MQT advisor of DB2 that returns a set of candidate MQT and their associated indexes. At the highest level, the DB2 Advisor works as a black-box view-index recommendation engine. The black-box has two inputs: a set of SQL statements known as the workload, and statistics describing the target database. There is only one output: the recommended views and indexes.
- Scheduled queries generator: the candidate MQT and indexes are then submitted to DMS that runs a genetic algorithm to find the best query order that produces the highest MQT benefit. The optimal solution requires the exploration of a search space of $N!$ permutation of the query workload. The objective of the genetic algorithm is establish a tradeoff between maximizing MQT cache hits, minimizing MQT materialization and minimizing base table accesses.
- Dynamic MQT management module is core of the approach. It uses a probabilistic model defining the usage and the materialization of any MQT candidate. The MQT pool (called cache) is managed by LRU that provides non-zero hit probability to the entire candidate MQT.

The main limitation of Phan et al.'s work: (i) it is DB2 DBMS dependent, (ii) certainly DB2 advisor takes into account the interaction among optimization structures (indexes, $\mathcal{MV}$, partitioning, and clustering) [27], but the interaction between queries is not well highlighted. Another aspect related to the query workload is the small number of the used queries. Nowadays, the number of queries may be very large (era of big queries). Note that the interaction between queries is crucial for selecting $\mathcal{MV}$ in different database deployment platforms [18, 19, 26], and (iii) having a genetic algorithm with fitness function using probabilistic parameters that require a predictable approach to get them is time consuming. To overcome these limitations, we propose a new scalable approach; called SLEMAS. This system plays the role of generic advisor.

The paper is organized as follows: Section 2 presents the Background. Section 3 details our contributions, where we present the scalable algorithm to capture the query interaction, $\mathcal{MV}$ selection algorithm, query scheduling and materialization strategy. Section 4 implements our results and validation in Oracle 11g. Section 5 concludes the paper by summarizing the main results and suggesting future work.

2. BACKGROUND

In this section, we present some concepts to facilitate the understanding of our approach that considers the interaction of queries to generate the view candidates.

2.1 Multi-Query Optimization

The Multi-query optimization (MQO) problem has well studied since 1980s [24, 22]. MQO tries to perform a batch of queries by exploiting some share common results. MQO problem can be divided into two phases [22]. The first phase is to prepare alternative plans for each query (Each query may be represented by an algebraic tree corresponding to its execution plan). These plans can identify common shared results among a set of queries. The second phase is to select exactly one plan for each query. As a result of second phase, the query plans may be merged to generate a graph structure called, *unified query plan* (UQP). The structure of this plan is similar to that proposed in [26]. The leaf nodes of the UQP represent the base tables. The root nodes represent the final query results and the intermediate nodes represent the common sub-expressions shared by the queries. Among intermediate nodes, we distinguish: (i) unary nodes representing the selection and the projection operations (ii) binary nodes representing join, union, intersection, etc. Figure 3a. presents an UPQ for a workload of 10 queries issue form star schema benchmark. Several unified query plans may exist for a given query workload due to the properties of relational algebra operations [26]. Sellis et al. [24] are proposed an $A*$ search to explore all possible UQP. But

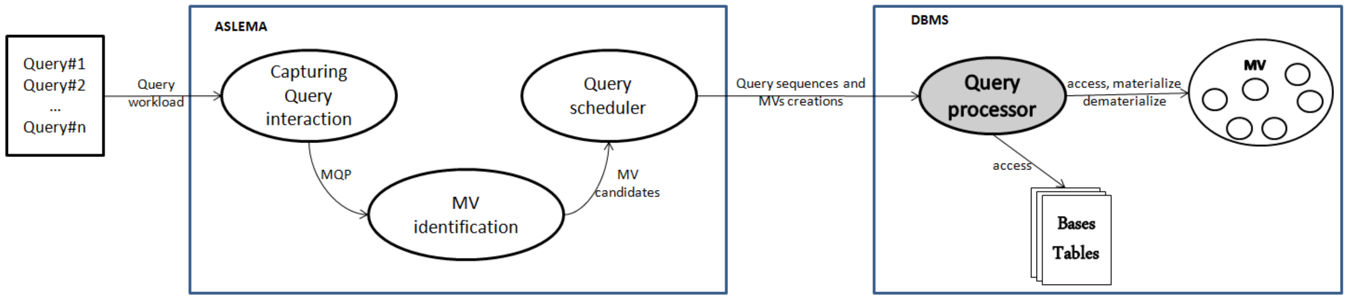


Figure 1: A global overview of our approach

exploring all possible UQP has been formulated as an NP-hard problem [24]. To reduce the number of exploring UQP many cost estimation functions are proposed [25]. Roy et al. [23] describe a greedy heuristic for generating a good alternative plans which maximize shared results. Dalvi et al. [5] extend the work of Roy et al.[23] by pipelining common shared results to maximize the benefit. With Big Queries Era, offering scalable algorithms for generating a best query plan becomes a crucial challenge [3].

2.2 Problem Formulation

Before formalizing the $\mathcal{MVP}$ considering the Query Scheduling Problem ($\mathcal{QSP}$), we think it would be wiser to propose a separate formalization of both $\mathcal{MVP}$ and $\mathcal{QSP}$. *Inputs:* (i) a relational data warehouse ($\mathcal{RDW}$), (ii) a workload with a set of queries represented by an UQP, (iii) a set of intermediate nodes candidates for materialization ; *Constraint:* a storage constraint $\mathcal{S}$; *Output:* a set of $\mathcal{MV}$ optimize the cost of processing $\mathcal{Q}$ and satisfying the constraint cost ($size(\mathcal{MV}) \leq \mathcal{S}$).

Similarly, the $\mathcal{QSP}$ is formalized as follows:
Inputs: (i) $\mathcal{RDW}$, (ii) a workload with a set of queries represented by an UQP, (iii) a set of intermediate nodes candidates for materialization; a disk allocation policy;
Output: scheduled queries of the workload into a new ordered set that the query having the least execution cost.

The $\mathcal{MVP}$ considering $\mathcal{QSP}$ takes (i) A $\mathcal{RDW}$ and (ii) a set of queries, (iii) a set of intermediate nodes candidates for materialization; a constraint representing the limited storage size. The problem aims at providing: (i) a scheduled set of queries and (ii) $\mathcal{MV}$, minimizing the overall processing cost of and satisfying the storage constraint. The search space of the combination of $\mathcal{MVP}$ and $\mathcal{QSP}$ becomes very huge [21]. In the next section, we propose a new approach supported by a tool (SLEMAS) dealing with this problem.

3. THE SLEMAS APPROACH

SLEMAS is a three-tier architecture as shown in the Figure 1: (1) an application tier representing the query workload, (2) SLEMAS with three main roles: (i) capturing of interaction among queries, (ii) generation of views candidate and (iii) query scheduling, and (3) a data storage tier that implements solutions recommended by SLEMAS. Its components are below detailed.

3.1 The Application Tier

This module receives from users a set of queries in a queue to be processed by a DBMS.

3.2 SLEMAS Tier

Once query workload is received, SLEMAS performs the following tasks:

3.2.1 *Capturing of interaction among queries:*

This step is performed by constructing the UQP. This construction has to take into account the Era of Big Queries. Usually, the interaction of queries is captured by the use of acyclic graph [26] (Figure 2.a. To respond to the context of Big Queries, we propose the use of hypergraph data structure that showed their efficiency in *Electronic Design Automation* (EDA) [13]. In our context the vertices and edges of the hypergraph represent respectively the join nodes and queries as shown in Figure 2-a. Since the number of nodes of our hypergraph can be very large, we adopt the same philosophy of EDA, where the hypergraph is partitioned according the interaction among queries. More precisely, each partition contains high interacted queries. To do the partitioning we use a tool widely used in EDA called *HMETIS* [13]. Figure 2-b gives an example of a such partitioning. Each partition is then transformed in a acyclic directed graph which is similar to traditional query acyclic graph. During this transformation, a cost driven approach to order the nodes is given [3]. This cost computes the processing cost of overall queries (Figure 2-c).

3.2.2 *Materialized Views Selection:*

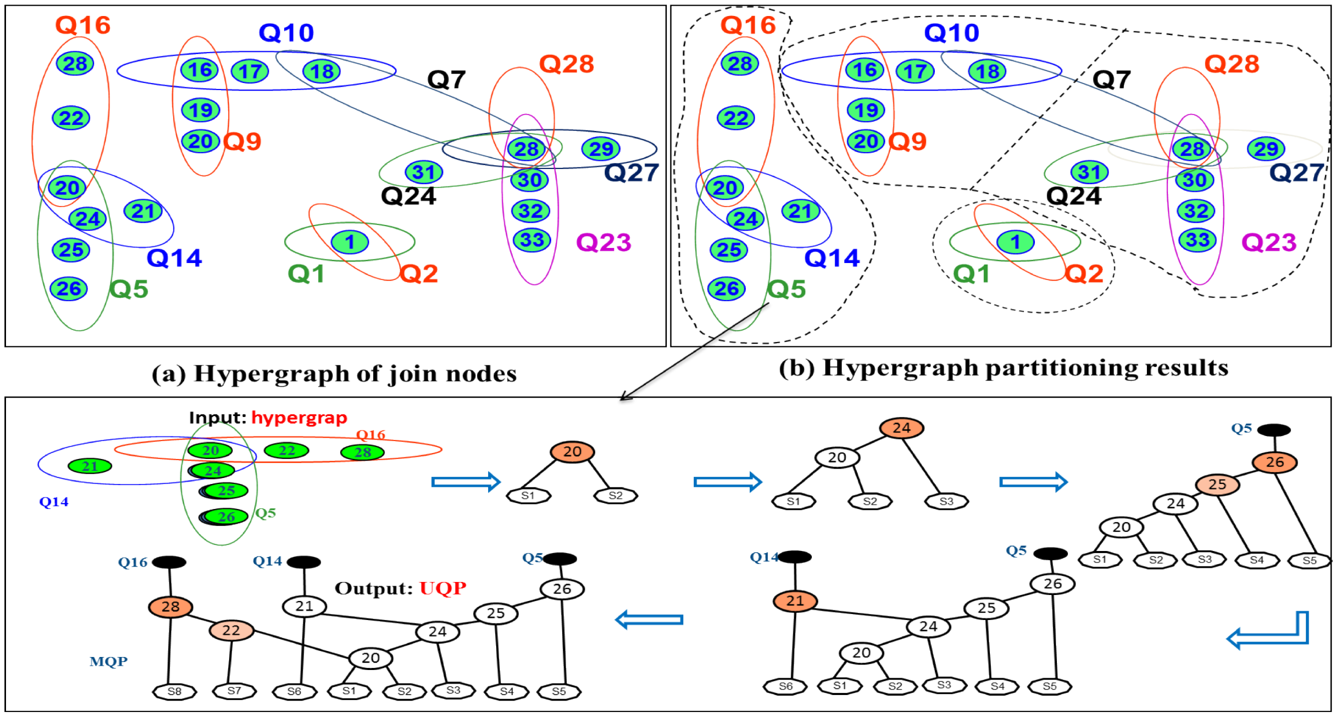
Note that all nodes of the global plan are candidate for materialization which may represent a huge number. For instance, in our experiments, we consider 1 000 queries involving 1552 join nodes. As a consequence a pruning mechanism is needed. It shall take into account the benefit of the nodes and their constraints related to their storage and maintenance. To do so, we define some functions:

- $cost_{WO}(q_i, \Phi)$: the processing cost of the query q_i without view(s).
- $cost_{WV}(q_i, V_j)$: the query processing cost of query q_i using the materialized view V_j.
- $cost_{Mat}(V_j)$: the maintenance cost of the view V_j.
- $Size(V_j)$: the cost needed to store the view V_j.

We define the benefit of a given view V_j (denoted by $Benefit(V_j)$) by:

$$benefit(V_j) = cost_{WO}(V_j) - cost_{WV}(V_j) - cost_{Mat}(V_j) \quad (1)$$

where $cost_{WO}(V_j)$ and $cost_{WV}(V_j)$ represent respectively the total processing cost of queries without/with the view



(a) Hypergraph of join nodes

(b) Hypergraph partitioning results

(c) Transformation of hypergraph to UQP

Figure 2: An Example of Query Interaction Capturing

V_j. Instead of treating the whole search space including all candidates as in the usual approaches for materializing views, we propose to use a *divide-conquer approach*, where the search space is divided into several sub search spaces, where each one corresponds to a connected component of the global graph (Figure 1). Contrary to the existing studies where they allocate the whole storage constraint to all views candidate, our approach allocates this storage to each component to be fair. Then, each component C_k is processed individually, where its nodes are sorted according their benefits and their ability to satisfy the storage constraint. Three selection cases are possible. Let N^{C_k} be the set of nodes of C_k having a greater benefit. The top nodes satisfying the storage constraint are selected.

3.2.3 *Query scheduling:*

To avoid massively view dropping, we schedule queries. This is done by respecting the following principle: *when a view is materialized, it should optimize the maximum of queries before its dropping.* Therefore, we propose the following procedure supported by an example in which we consider a connected component with 14 queries (Figure 3-a).

1. The queries are grouped in many distinct components which the interaction (sharing of intermediate results), is important between queries inside component and negligible interaction between components. Our scheduler aims to schedule the queries in each component and the order between components.

2. The scheduler is based on maximizing the benefit of reusing nodes, so the order is guided by nodes. The materialization of a node prompts that the queries which use this node are the following to be run. So, no consideration of space constraint in query scheduler.

3. The identification of node(s) of each component with maximal benefit (called queen nodes). In our example, four *queen nodes* are selected: $\{qn_1, qn_2, qn_3, qn_4\}$ (represented by solid nodes in Figure 3-a).

4. Ordering queen nodes: Let N^{C_k} be the number of

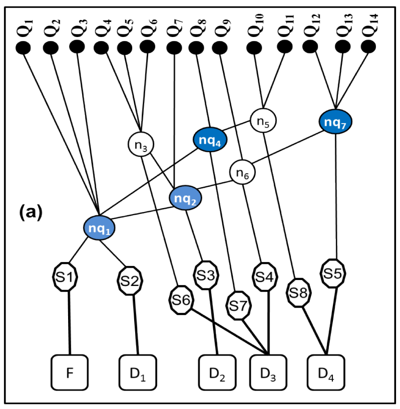
(b)

Node	Benefit
nq$_1$	400
nq$_2$	100
nq$_4$	300
nq$_7$	200

Node	Benefit
nq$_1$	400
nq$_4$	300
nq$_7$	200
nq$_2$	100

(c)

Query	Use	Weight
Q$_1$	nq$_1$	400
Q$_2$	nq$_1$	400
Q$_3$	nq$_1$	400
Q$_4$	nq$_1$, nq$_2$	500
Q$_5$	nq$_1$, nq$_2$	500
Q$_6$	nq$_1$, nq$_2$	500
Q$_7$	nq$_1$, nq$_2$	500
Q$_8$	nq$_1$, nq$_4$	700
Q$_9$	nq$_1$, nq$_2$	600
Q$_{10}$	nq$_1$, nq$_4$	700
Q$_{11}$	nq$_1$, nq$_4$	700
Q$_{12}$	nq$_1$, nq$_2$, nq$_7$	700
Q$_{13}$	nq$_1$, nq$_2$, nq$_7$	700
Q$_{14}$	nq$_1$, nq$_2$, nq$_7$	700



(d)

Query order	Q$_8$	Q$_{10}$	Q$_{11}$	Q$_{12}$	Q$_{13}$	Q$_{14}$	Q$_9$	Q$_4$	Q$_5$	Q$_6$	Q$_7$	Q$_1$	Q$_2$	Q$_3$

Figure 3: Example of our Scheduling Approach

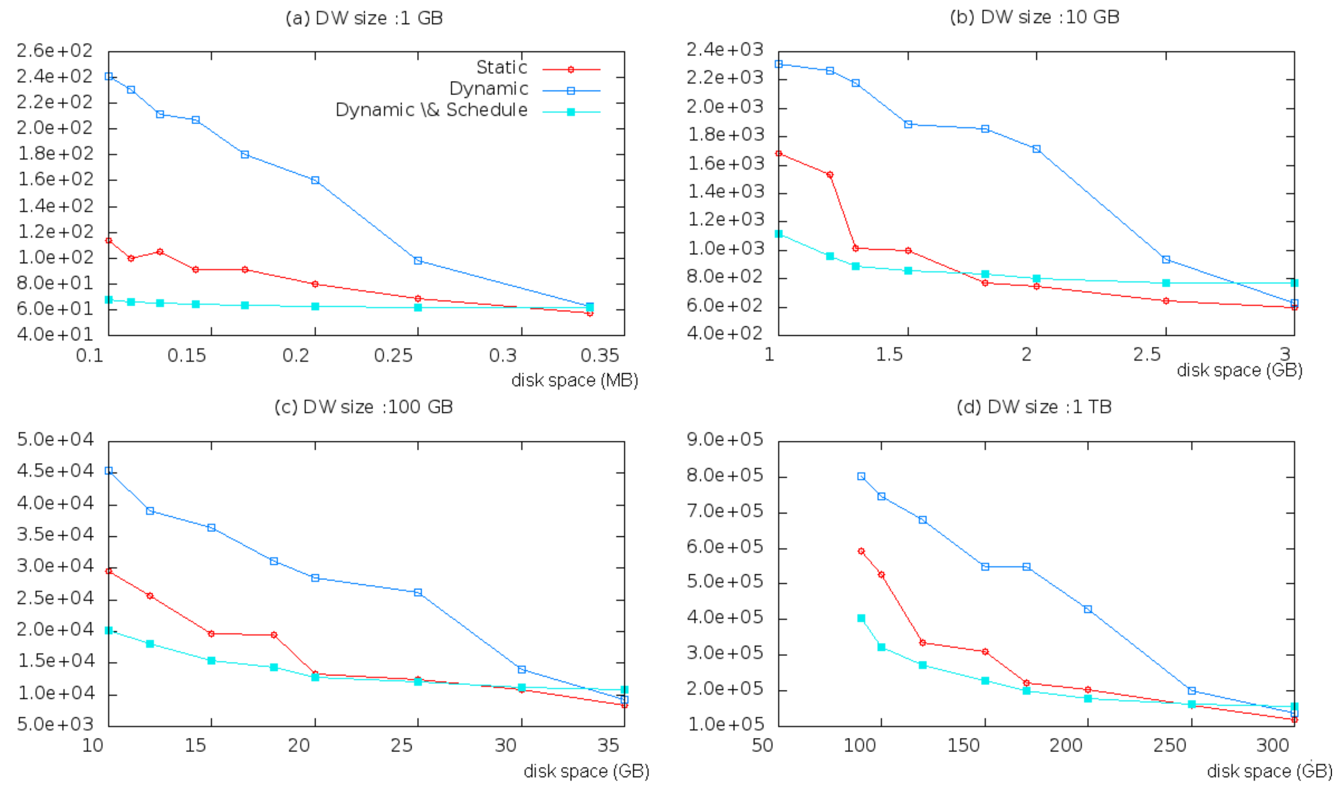


Figure 4: Advantage of dynamic materialization with query scheduling

nodes of the component C_k. Their ordering is based on their benefit. The benefit of the queen nodes are propagated to their queries. As a consequence, each query may be assigned to a weight representing the sum of the benefit of its nodes. These weight are then used to schedule the queries based on their overall benefit (Figure 3-d). The query scheduling process not consider the constraint space

Till now, materialized views candidate are identified and the order of queries. Based on the different cost models available at SLEMAS side, it can easily decide on materializing or dematerializing views by performing simulation using algorithm 1.

Algorithm 1 materializeView (mv)

```
1: $cost \leftarrow estimateMaintenance(mv)$;{Estimate the maintenance cost of the view using the generic cost model}
2: changeStat($mv$, true);{ change the stat of the view as materialized}
3: $diskSpace \leftarrow diskSpace - sizeOf(mv)$;
4: return $cost$;
```

3.3 Data Storage Tier

The results obtained by SLEMAS are translated respecting the target DBMS then receives a creation and dropping scripts of materialized views and the pre-ordered queries.

4. EXPERIMENTAL EVALUATION AND ANALYSIS

We conduct several experiments to evaluate the efficiency of SLEMAS. First of all, we develop a simulation tool using Java. It allows to get automatically the characteristics of the meta data of the data warehouse and run the three algorithms: (1) SLEMAS's approach; (2) dynamic materialization algorithm proposed by Phan et al's [21] and (3) $\mathcal{MV}$ selection algorithm proposed by Yang et al's [26] using a static formalization. To analyze the behaviors of these algorithms, we consider four scenarios: (1) using static materialization, (2) dynamic materialization, (3) with considering query scheduling and (4) without query scheduling. These scenarios are tested by varying: (a) the size of data warehouse, (b) consideration of different query workload's randomly generated using SSB query generator and (c) varying the storage space constraint. The simulated results are then deployed on Oracle 11g *DBMS*, running on a Core 2 Duo server with 2.40GHZ CPU and 32 GB of main memory. The star Schema Benchmark (SSB) with 100 GB of data containing a fact table *Lineorder* and 4 dimension tables [20]: *Part*, *Customer*, *Supplier* and *Date* is used.

In the first experiments, we test the interest of dynamic and query scheduling on optimizing queries. To perform our experiments we consider a data warehouse with different sizes (1Gb, 10Gb, 100Gb and 1 Tb) and a workload of 30 queries, the candidate nodes are selected using our approach. Three scenarios are considered: (i) naive scenario in which nodes are materialized till the saturation of storage space,

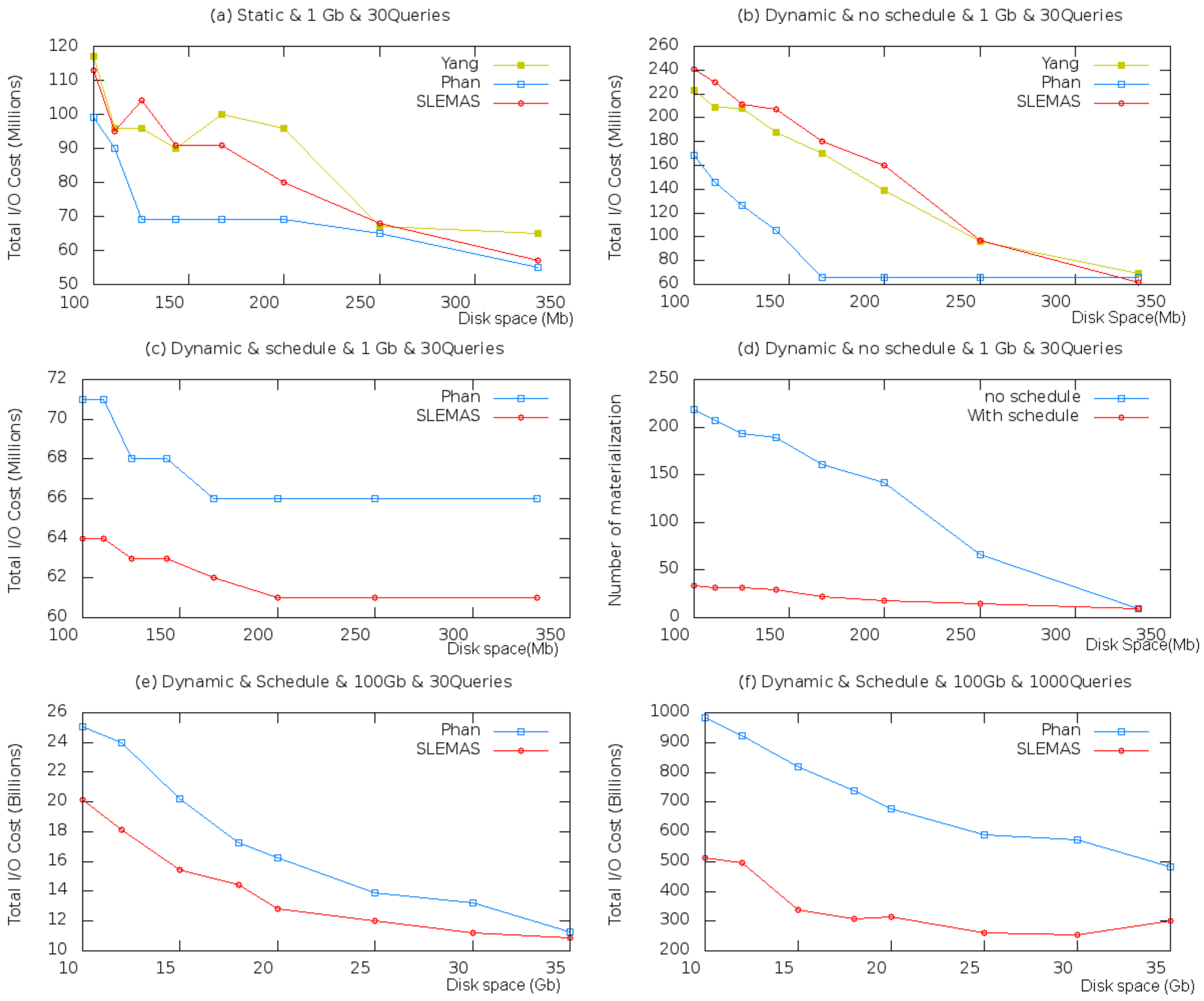


Figure 5: Performance of SLEMAS's approach

(ii) materializing without scheduling using our dynamic approach and considering the workload as pre-ordered[2] and (iii) materializing with query scheduling. The overall cost of queries in terms of inputs/outputs is then computed by varying the storage constraint. Figure 4 summarizes the obtained results. The main lesson is: the dynamic materialization with query scheduling outperforms the other scenarios whatever the size of the data warehouse. This shows the interest of incorporating the query scheduling in materializing views.

In the second experiments: we test the performance of our approach compared with two existing approaches: Phan et al.'s method [21], and Yang et al.'s method [26]. For Phan method, we have developed the following algorithms: (i) a genetic algorithm to find an optimal query permutation by emulating Darwinian natural selection of 1000 generations; (ii) algorithm to select candidate nodes which are the nodes that have greater benefit have been selected as candidates (In Phan et al.'s, they are used DB2 advisor to have those all nodes with greater benefit); (iii) pruning algorithm of candidate views using their benefit; (iv) an evaluation algorithm to estimate the total net benefit of using pruned candidate views set by query workload; (v) algorithm to manage the cache of the views (LRU). For Yang, we have developed the following functions: (i) generation of individual plan tree, (ii) generating Multiple Views Processing Plans (*MVPP*), using merging individual plans (iii) selecting materialized views using *MVPP* (iv) estimation of query MVPP using views. To show the performance of our approach three scenarios are considered:

1. **Static materialization:** we consider a data warehouse with 1Gb and a workload of 30 queries, the nodes selected by each algorithms are materialized till the saturation of storage space. As shown in the Fig-

[2]The query scheduling module of our approach in this case is obsolete.

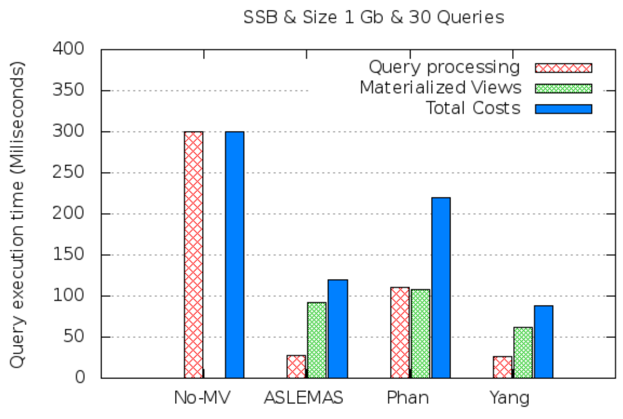


Figure 6: Oracle validation on data warehouse of 1Gb

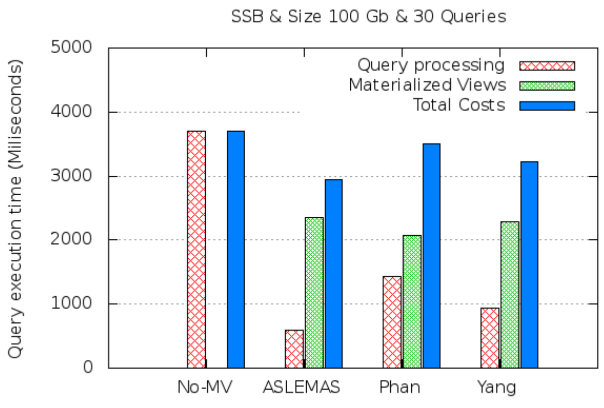


Figure 7: Oracle validation on data warehouse of 100Gb

ure 5-a, there is not a big difference between the three methods, which improve that our approach not avoids the selection of best views.

2. **Dynamic materialization without query scheduling**, we have used the same configuration as static materialization. As shown in the Figure 5-b, we show that Phan is more better than our approach. This is due to the difference of materialization/dematerialization number (Figure 5-c). Phan et al.'s method tries to find best candidate views that optimize the workload globally which minimize the dropping of the views. But the views in our approach a divided a many sub sets which each sub-set optimize some queries, which increase the probability of dropping views if the query not scheduled.

3. **Dynamic materialization with query scheduling:** we have used data warehouse with different size (1Gb and 100Gb) and a workload of 30 queries. As shown in the Figures 5-c and 5-e, our approach outperforms Phan method because the number dropping is minimal and each materialized views are used maximally to optimize the queries of the component. As shown in the Figures 5-f, our approach performs more when we use big queries (in our tests: 1000 queries).

4.1 Validation in Oracle 11g

ue to the complexity and time needed to deploy all theoretical solutions on Oracle DBMS, we propose the following: we consider a workload of 30 queries running on two data warehouses (1GB and 100GB). The disk storage is set to 400MB (39%) for the first dataset(Figure 6) and 30GB (30%) for the second dataset (Figure 7).

The obtained results described in the Figures 6 and 7 which are quite similar to those obtained by our simulator. This shows the quality of our used cost models.

5. CONCLUSION

In this paper, we addressed an important problem which is the dynamic materialized view selection by considering the query scheduling. Both problems are hard. To solve this problem, we propose a methodology supported by a generic advisor called SLEMAS. It is 3-tiers architecture included: (1) an application tier representing the query workload, (2) SLEMAS with three main components: (i) capturing of interaction among queries. To offer a scalable algorithm, we proposed the use of hypergraph largely used in the EDA domain. (ii) Generation of views candidate performed by divide-conquer approach, in which the hypergraph is partitioned into several connected components. (iii) Query scheduling allows re-ordering the query workload based on their benefit in using materialized views. (3) A data storage tier that implements solutions recommended by SLEMAS. Our approach is compared against the most important state of art works and the obtained results show the efficiency and effectiveness of our approach.

Currently, we are working in two directions: the incorporation of another optimization structure which is the indexes to SLEMAS and development of a more sophisticated tool with nice interfaces and offering several API to connect different DBMS platforms.

6. REFERENCES

[1] L. Bellatreche, A. Kerkad, S. Breß, and D. Geniet. Roupar: Routinely and mixed query-driven approach for data partitioning. In *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, pages 309–326. Springer, 2013.

[2] A. G. Bello, K. Dias, A. Downing, J. Feenan, J. Finnerty, W. D. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in oracle. In *VLDB*, pages 659–664, 1998.

[3] A. Boukorca, L. Bellatreche, S.-A. B. Senouci, and Z. Faget. Sonic: Scalable multi-query optimization through integrated circuits. In *Database and Expert Systems Applications*, pages 278–292. Springer, 2013.

[4] L. W. F. Chaves, E. Buchmann, F. Hueske, and K. Böhm. Towards materialized view selection for distributed databases. In *EDBT*, pages 1088–1099, 2009.

[5] N. N. Dalvi, S. K. Sanghai, P. Roy, and S. Sudarshan. Pipelining in multi-query optimization. *Journal of Computer and System Sciences*, 66(4):728–762, 2003.

[6] N. Daneshpour and A. A. Barforoush. Dynamic view management system for query prediction to view

materialization. *IJDWM*, 7(2):67–96, 2011.

[7] A. Diwan, S. Sudarshan, and D. Thomas. Scheduling and caching in multi-query optimization. In *International Conference on Management of Data COMAD, Delhi, India*, 2006.

[8] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[9] A. Gupta, S. Sudarshan, and S. Viswanathan. Query scheduling in multi query optimization. In *IDEAS*, pages 11–19, 2001.

[10] H. Gupta. *Selection and maintenance of views in a data warehouse*. Ph.d thesis, Stanford University - USA., 1999.

[11] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, pages 205–216, 1996.

[12] V. Kantere, D. Dash, G. François, S. Kyriakopoulou, and A. Ailamaki. Optimal service pricing for a cloud cache. *IEEE TKDE*, 23(9):1345–1358, 2011.

[13] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. In *ACM/IEEE Design Automation Conference (DAC)*, pages 343–348, New York, NY, USA, 1999. ACM.

[14] Y. Kotidis and N. Roussopoulos. Dynamat: a dynamic view management system for data warehouses. *SIGMOD Rec.*, 28(2):371–382, June 1999.

[15] M. Lawrence. Multiobjective genetic algorithms for materialized view selection in olap data warehouses. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 699–706. ACM, 2006.

[16] C. Maier, D. Dash, I. Alagiannis, A. Ailamaki, and T. Heinis. Parinda: an interactive physical designer for postgresql. In *EDBT*, pages 701–704, 2010.

[17] I. Mami and Z. Bellahsene. A survey of view selection methods. *SIGMOD Record*, 41(1):20–29, 2012.

[18] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *SIGMOD*, pages 307–318, 2001.

[19] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas. Mrshare: Sharing across multiple queries in mapreduce. *PVLDB*, 3(1):494–505, 2010.

[20] X. C. Pat O'Neil, Betty O'Neil. Star schema benchmark. June 2009.

[21] T. Phan and W.-S. Li. Dynamic materialization of query views for data warehouse workloads. In *ICDE*, pages 436–445, 2008.

[22] A. Rosenthal and U. S. Chakravarthy. Anatomy of a mudular multiple query optimizer. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 230–239, 2006.

[23] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. *ACM SIGMOD Record*, 29(2):249–260, 2000.

[24] T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems*, 13(1):23–52, March 1988.

[25] K. Shim, T. Sellis, and D. Nau. Improvements on a heuristic algorithm for multiple-query optimization. *Data & Knowledge Engineering*, 12(2):197–222, 1994.

[26] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *VLDB*, pages 136–145, 1997.

[27] D. C. Zilio, J. Rao, S. Lightstone, G. M. Lohman, A. J. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. In *VLDB*, pages 1087–1097, 2004.