Cache Pattern with Multi-Queries

Nicoleta Liviana TUDOR
The Petroleum-Gas University of Ploiesti
B-dul Bucuresti, nr. 39 , RO-100680, Ploiesti
ltudor@upg-ploiesti.ro

***Abstract*—This article proposes a cache pattern with multi-queries and describes the multi-query optimization with scheduling, caching and pipelining**

A set of cache patterns is derived from a set of class of multi-queries that are loaded into the cache. Each cache pattern represents a unique equivalence class in the set of patterns.

The multi-query optimization with scheduling, caching and pipelining provides efficient heuristics, for a good queries ordering using a single invocation on the entire batch of queries. Multi-query optimization chooses the results of sub-expression that should be admitted to or discarded from cache, when it executes queries.

We introduce the heuristic of pair queries and define the equivalence class of multi-queries from cache pattern. We show that the union of all equivalence classes of queries from the cache patterns is the set of cache patterns.

***Index Terms*—equivalence class of queries from the cache, multiple queries optimization, multi-query caching, set of cache patterns, the heuristic of pair queries**

I. Introduction

This article proposes a cache pattern with multi-queries and describes the multi-query optimization with scheduling, caching and pipelining. The paper is organized as follows:

- the section *Prior work* presents some improved alternatives of multi-queries optimization with scheduling and caching
- the section *Algebraic Pattern for Multi-Queries Caching* provides a mathematical description of the query caching model. It defines CP-MQ notation as the set of patterns derived from a set of class of multi-queries, upon a set of relations and it defines the concept relational scheme of cache pattern with multi-queries. In this section, we demonstrate some properties of equivalence classes defined on the set of cache patterns
- the section *Multi-Queries Optimization* intends to find an optimal permutation of the queries such that the computation cost of the set is minimum. The multi-queries optimization chooses equivalent query expressions from the possible semantic rewritings of the input query. In this section, we use a DAG representation of queries, which can be applied to System R style optimization, we introduce the heuristic of pair queries. We define the equivalence class of multi-queries from cache pattern and we determine the union of all equivalence classes of multi-queries from the set of cache patterns.

II. Prior Work

Database systems frequently have to execute a batch of multiple queries, which may contain several common subexpressions. Current approaches to multi-query optimization (MQO) assume there are three problems: query scheduling, caching and pipelining. The scheduling is the problem of finding the best order of evaluation of expressions and the caching *is* the problem of deciding when to store a shared result in cache, and when to discard it, to minimize evaluation cost under cache space constraints [1].

There has been a significant recent work on multi-queries optimization. [2] demonstrates the practical applicability of multi-query optimization (MQO) based on efficient algorithms for implementing a greedy heuristic.

Multiple queries optimization (MQO) has been expressed in several contexts in the recent past including transient views [3], view maintenance [4], XML query optimization [5] and continuous query optimization [6].

According to Sellis [7], MQO provides one solution to view selection. The goal of this method is to exploit shared data between a set of queries or views and identify additional views for transient or permanent materialization, that are used to share intermediate results and improve performance. Multi-query optimization has been successfully applied in view maintenance [4] and in the optimization of inter-query execution [8]. However, evaluating shared sub-expressions increases the complexity of query optimization. Query caching decreases optimization cost because each query is evaluated against a single prototype.

Wang et al. [9] propose the SkyQuery workload-driven technique for choosing the logical unit of cache replacement that is adaptive and self-organizing. This prototype is a combination of attributes that is accessed by the same class of queries.

Gupta [10] studied scheduling and caching in MQO, and presented results on intractability of the caching problem, and approximation algorithms for special cases of the caching problem. They provide approximation techniques for caching given a fixed schedule, and they only provide heuristics for finding good schedules; moreover, their heuristics require a large number of MQO invocations to decide a good schedule and is seen to add a very large overhead even for very small query batches.

Other approaches to multi-query optimization assume there is infinite disk space, and very limited memory space. Pipelining was the only option considered for avoiding expensive disk writes. [11] provides algorithms for the problem of pipelining in MQO, but does not consider the problems of scheduling and caching. The availability of fairly large and inexpensive main memory motivates the need to make best use of available main memory by for caching shared results, and scheduling queries in a manner that facilitates caching.

The problem of caching shared results was also addressed by Tan and Lu [12]; they provide heuristics for scheduling

Digital Object Identifier 10.4316/AECE.2010.02014

in MQO, but they require a pairwise test for deciding a good order of queries.

In [13], [14] an active query-caching framework for form-based proxy caching of database backed websites is described and significant gains compared to passive query caching are reported. Lightweight Directory Access Protocol (LDAP) replication has widely been used for improving performance, scalability and availability of directory based web applications. An LDAP caching solution, which provides significant query hit ratio, while caching only a small fraction of the records in the directory is thus desirable. There are two distinct problems associated with LDAP query caching [15]:

(i) Determining whether an LDAP query is contained in another query (query containment),

(ii) Using caching algorithms which make efficient caching, prefetching, cache replacement decisions to maximize the fraction of queries answered from the cache. [16] introduce the notion of generalized queries and propose a *caching* algorithms.

In the next section, we define an algebraic pattern for multi-queries caching, where each cache pattern represents a unique equivalence class in the set of patterns.

III. Algebraic Pattern for Multi-Queries Caching

This section provides a mathematical description of the query caching model. The method for specifying query pattern has as input a set of queries Q and outputs a set of patterns P, which serve as the cache pattern. Each query is matched exactly one pattern, whereas each pattern is derived from a set of related queries.

To define the cache pattern with multi-queries, the following shall be considered:

- a set of domains $D_i = (D_{i1}, D_{i2}, \ldots, D_{ij, \ldots})$, D_{ij} – range of values, $i = 1, \ldots, n$, $j = 1, \ldots, m$, where $m,n \in N^*$
- a set R of n relations $R_1, \ldots, R_n$ and each relation consists of a set of attributes:

$R_1(c_{11}, c_{12}, \ldots, c_{1i}, \ldots)$, c_{1i} attribute, $c_{1i} \in D_1$, $i = 1, \ldots, n$,
$R_2(c_{21}, c_{22}, \ldots, c_{2i}, \ldots)$, c_{2i} attribute, $c_{2i} \in D_2$, $i = 1, \ldots, n$, ...
$R_n(c_{n1}, c_{n2}, \ldots, c_{ni}, \ldots)$, c_{ni} attribute, $c_{ni} \in D_n$, $i = 1, \ldots, n$,
where $\max(|R_i|)_{i=1,\ldots,n} <= m$

- the set Q of all queries in the cache pattern, in which $Q_i \in Q$ is the i^{th} query in the cache pattern

Papadomanolakis [17] introduced the concept of a Query Access Set (QAS), which is the subset of attributes from a single relation in R that are referenced by a query in Q. For query prototypes, Wang et al. [10] redefine Query Access Set to be the set of attributes from every relations in R that are referenced by a query in Q.

Further we introduce CP-MQ notation and define the concept relational scheme of cache pattern with multi-queries.

Definition 1: A cache pattern with multi-queries, named CP-MQ is the set of cache patterns derived from a set of class of multi-queries, upon a set of relations and loaded into the cache.

Definition 2: The relational scheme of cache pattern with multi-queries (CP-MQ) is defined as the set of attributes from a set R of n relations that are referenced by a multi-query and are loaded into the cache. So, it is noticing:

CP-MQ (P_k, Q_i, C_{it}) = { C_{it} attributes loaded into the cache pattern $P_k \in P$ and referenced by $Q_i \in Q$, $i, j = 1, \ldots n$, $k >=1$ }

Now, we define an equivalence relation on a set of queries. Let us consider two queries Q_i and Q_j on relations $R_1, R_2, \ldots, R_n$ and an implementation of the JOIN operator defined as follows:

Q_i = JOIN $(R_k, R_l, R_k.c_{kl} = R_l.c_{ll})$, $k_1, l_1 = 1, \ldots, n$, $i>=1$

Let be two queries $Q_i, Q_j \in Q$. Then a subset $\rho \subset Q_i \times Q_j$ is a binary relation between Q_i and Q_j [18].

Definition 3. A binary relation ρ on set of queries Q is named equivalence relation if:

i. $Q_i \ \rho \ Q_i$
ii. $Q_i \ \rho \ Q_j \Rightarrow Q_j \ \rho \ Q_i$
iii. $Q_i \ \rho \ Q_j$ and $Q_j \ \rho \ Q_k \Rightarrow Q_i \ \rho \ Q_k$, $\forall \ Q_i, Q_j, Q_k \in Q$

Let us consider that the equivalence relation ρ on the set Q of queries loaded into the cache pattern $P_{k\ (k>=1)}$, is defined as follows:

$Q_i \ \rho \ Q_j \Leftrightarrow$ CP-MQ (P_k, Q_i, C_{it}) = CP-MQ (P_k, Q_j, C_{js}), that means Q_i and Q_j access the set of attributes, that are loaded into the same cache pattern $P_{k\ (k>=1)} \in P$.

Definition 4. The equivalence class of query Q_i is defined as a set:

$[Q_i] = \{ Q_j \in Q / Q_i \ \rho \ Q_j \}$

Definition 5. The equivalence class of query from the cache pattern $P_{k\ (k>=1)}$ is the set:

$[Q_i]$ = { $Q_j \in Q$ / CP-MQ (P_k, Q_i, C_{ik}) = CP-MQ (P_k, Q_j, C_{js}), $P_{k\ (k>=1)} \in P$ }

Observation: Each cache pattern $P_{k\ (k>=1)}$ represents a unique equivalence class in the set of patterns P. The set of attributes referenced by queries in P_k are loaded into the cache as one unit.

Further we present some properties of equivalence classes of queries loaded into the cache.

Proposition 1. Let be a cache pattern (P, Q), where P is a set of cache patterns, $P \neq \phi$, Q is a set of queries loaded into the cache and let ρ be an equivalence class on P. Then the equivalence classes on P have the properties:

i). $Q_i \in [Q_i]$, $\forall \ Q_i \in Q$ set of queries loaded into the cache.
In particular, $[Q_i] \neq \phi$

ii). $[Q_i] = [Q_j] \Leftrightarrow Q_i \ \rho \ Q_j$, where Qi and Qj are related queries

iii). $[Q_i]$ and $[Q_j]$ are equivalence classes $\Rightarrow$ $[Q_i] = [Q_j]$ or $[Q_i] \cap [Q_j] = \phi$

Proof:

i). $Q_i \ \rho \ Q_i \Rightarrow Q_i \in [Q_i]$, $\forall \ Q_i \in Q$

ii). Let us demonstrate "$\Rightarrow$": let us suppose that $[Q_i] = [Q_j]$

We have $Q_i \in [Q_i]$ then $Q_i \in [Q_j] \Rightarrow Q_i \ \rho \ Q_j$

Let us prove "$\Leftarrow$": let us suppose that $Q_i \ \rho \ Q_j$ and Qi and Qj are related queries

We must demonstrate that $[Q_i] \subset [Q_j]$ [1]

Let x be, $x \in [Q_i]$ then $x \ \rho \ Q_i$, but ρ is transitive then

$x \rho Q_j \Rightarrow x \in Q_j \Rightarrow [Q_i] \subset [Q_j]$

Let us demonstrate that $[Q_j] \subset [Q_i]$ [2]

Let x be, $x \in [Q_j]$ then $x \rho Q_j$, but ρ is transitive then $x \rho Q_i \Rightarrow x \in Q_i \Rightarrow [Q_j] \subset [Q_i]$

[1] and [2] $\Rightarrow [Q_i] = [Q_j]$

Let us assume that $[Q_i] \cap [Q_j] \neq \phi \Rightarrow (\exists)\, x \in [Q_i] \cap [Q_j] \Rightarrow x \rho Q_i$ and $x \rho Q_j$

but ρ is symmetric $\Rightarrow Q_i \rho x \Rightarrow Q_i \rho Q_j \Rightarrow$ (from ii) $[Q_i] = [Q_j]$.

IV. Multi-Queries Optimization

Database systems are often required to execute a batch of queries, which may contain several common sub-expressions.

Problem: Consider two queries: $Q_1: (R \rhd\lhd S) \rhd\lhd T$ and $Q_2: (R \rhd\lhd S) \rhd\lhd U$. If we evaluate the two queries separately, then we must recompute $R \rhd\lhd S$. If we materialize the result of $R \rhd\lhd S$, although we do not have to recompute the result, we have to compute the additional cost of writing and reading the result of the shared expression. Thus, results would be shared only if the cost of recomputation is higher than the cost of materialization and reading. If we pipeline the results of $R \rhd\lhd S$ to both the queries, we do not have to recompute the result of $R \rhd\lhd S$ and we also save the costs of materializing and reading the common expression. But, if all the operators are pipelined, then the schedule may not be realizable.

The problem of scheduling, caching and pipelining in multi-query optimization has two main aspects:

- choosing the results of sub-expression that should be admitted to or discarded from cache, as it executes queries and
- choosing a good order of queries.

V. Equivalent Query Expressions

The cost of multi-queries optimization depends on the number of expressions equivalent for a given expression. For the general case, with an arbitrary set operators, for query order transformation, the number of equivalent rewritings of a query expression is exponential in the size of original query expression [1], [6]. The Volcano optimization algorithm generates all possible semantic transformations of the input query.

The Logical Query DAG (LQDAG) is the space of all possible equivalent relational algebra expressions. Diwan et al. [1] consider that a given expression can have infinitely many equivalent expressions generated by transformation rules. To avoid this problem they assume that given a query of size n, a query transformation rule can introduce at most a constant number of new operators or constant values and transformation rules will only generate expressions bounded in size by a polynomial p(n). This ensures that only a limited number of expressions can be generated.

For a query, the equivalent transformations are executed, the physical operators are chosen, and the indexes are built. For example, for the JOIN operator ($\rhd\lhd$), the transformations of the query are:

$$R \rhd\lhd S = S \rhd\lhd R$$
$$(R \rhd\lhd S) \rhd\lhd T = R \rhd\lhd (S \rhd\lhd T)$$

The transitivity can be represented using trees as follows (figure 1):

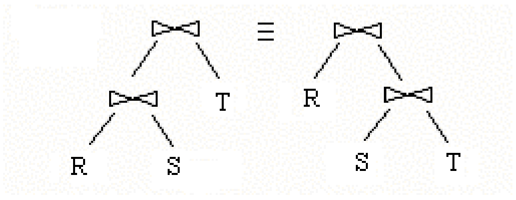

Figure 1. Tree representation of the transitivity.

If the join operation contains a projection (π), and x is a subset of attributes of R, y is a subset of attributes from the relation S and z is the intersection of the attributes from R and from S that are used by the p predicate, then the transformations of the JOIN can be:

$$\pi_{xy}[\sigma_p(R \rhd\lhd S)] = \pi_{xy}\{\sigma_p[\pi_{xz}(R) \rhd\lhd \pi_{yz}(S)]\}$$

The Logical Query DAG space can be represented as in figure 2:

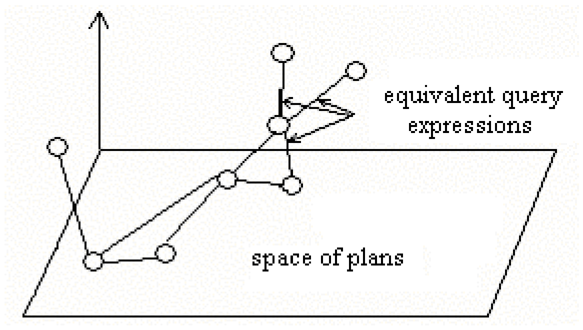

Figure 2. Logical Query DAG space.

VI. Optimal Order Of Multi-Queries

In this section, we intend to provide efficient heuristics for scheduling, caching and pipelining in multi-queries optimization, for a good query ordering using a single invocation on the entire batch of queries. We use a DAG representation of queries, which can be applied to System R style optimization.

Find an optimal permutation of the queries such that the computation cost of the set is minimum means that every two adjacent queries in the permutation have a common relation. Diwan et al. [1] show that this problem is equivalent to finding a dominating trail of a graph and this is equivalent to finding a Hamiltonian path in a cubic subgraph. They proposed heuristics for query scheduling, which provides a good query ordering. As an heuristic, the problem of scheduling is considered as the problem of finding the maximum weight Hamiltonian path in the QuerySet Graph, as a graph with nodes as queries belonging to the set of queries and node Qi connected to node Qj with an edge of weight Benefit(Qi, Qj). Benefit(Qi, Qj) is the benefit gained by executing both the queries as a batch using a cache space restricted to C and it is defined as:

Benefit(Qi,Qj) = cost(Qi) + cost(Qj) - MQO cost(Qi,Qj), where C is the cache size. This heuristic requires calculation of the benefit for each combination, and results in many calls MQO, and is thus expensive and impractical. It is known that query scheduling and caching problem is NPcomplete.

The Volcano representation of query plans, outlined in [19] and [2], uses all possible semantic rewritings of the input query. The Logical Query DAG (LQDAG) is an AND-OR DAG, representing the space of all possible equivalent relational algebra expressions. The Physical Query DAG (PQDAG) is used to specify the various algorithms available to evaluate a relational algebra expression and also the physical properties that are satisfied.

Multi-query optimizers generate query execution plans with common subexpressions used more than once, and thus nodes in the plan may have more than one parent. The Plan DAG is a DAG structured query plan.

Flum, Frick, Grohe define the DAG graph [20]:

Definition 6. The DAG graph is a directed graph with vertex set Q^I and an edge from v to w if w occurs in f(v), where:

$I = (Q^I, f^I, root^I, Q^I_1, ..., Q^I_k)$, Q^I is the (finite) set of vertices,

$f^I : Q^I \rightarrow (Q^I)^k$ is a function whose graph is acyclic and has vertices of in-degree 0, $root^I$ is the vertex of in-degree 0, and $Q^I_1, ..., Q^I_k$ are subsets of Q^I

For a query Q_i of size n, it knows the number of distinct plans DAG is exponential in n. Feasibility of pipelining of arbitrary subset of edges can be tested in polynomial time by searching for C-cycles [12]. Finding the optimal scheduling and caching strategy for a fixed set of pipelined edges in a fixed plan DAG is reduced to a minimum cost path problem.

Let us consider a set of cache patterns P and a set R of n relations $R = \{ R_1, ..., R_n\}$, $n \in N^*$. Let be $P_{k\ (k>=1)} \in P$ a cache pattern defined as follows:

- a query $Q_i \in Q$, on relations R_i, R_j, Q_i loaded into the cache from pattern $P_{k\ (k>=1)} \in P$
- a cache size C
- an implementation of the JOIN operator defined as follows:

$Q_i = JOIN (R_i, R_j, R_i.c_{ik} = R_j.c_{jp})$, $R = \{ R_1, ..., R_n \}$, $n \in N^*$, $i, j, i_k, j_p = 1, …, n$ and each Q_i is defined as a pair (evaluated query, query in cache) and the total size of the query results in cache is less than the cache size C.

Let be a graph $G = (Q, U)$, where $Q \neq \phi$ and

$U = \{ u = [Q_i, Q_t] /\ Q_i, Q_t \in Q, Q_i = JOIN (R_i, R_j, R_i.c_{ik} = R_j.c_{jp}), i, t, i_k, j_p = 1, …, n\}$

Further we introduce the heuristic of pair queries and define the equivalence class of multi-queries from cache pattern.

Definition 7. We will note $h (Q_i, Q_j)$ the heuristic of pair queries (Q_i, Q_j), where $Q_i, Q_j \in Q$, the maximum degree of vertices within the path of pipelined edges from Q_i to Q_j:

$h (Q_i, Q_j) = \max_{k=i}^{j}(degree_p(Q_k))$, where $Q_k \in$ the path of pipelined edges from Q_i to Q_j and $degree_p (Q_k)$ is the degree of node Q_k, if we only take into account the adjacent pipelined edges.

For example, the figure 3 shows a query graph which contains a fixed set of pipelined edges, in a fixed plan DAG where solid edges are pipelined and dotted edges are materialized.

$h (Q_i, Q_j) = \max \{degree (Q_i), degree (Q_k), degree (Q_j)\} = \max \{ 1, 4, 1 \} = 4$

$h (Q_p, Q_d) = \max\{degree (Q_p), degree (Q_d) \} = \max\{ 1, 2\} = 2$

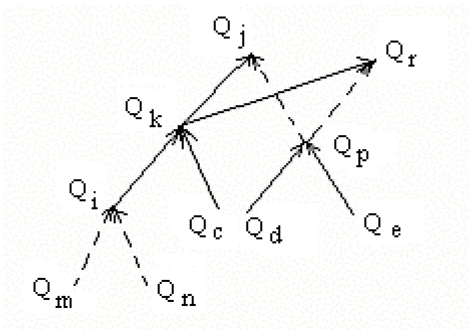

Figure 3. The query graph for a plan DAG.

Let us consider that the equivalence relation δ on the P, with a set Q of queries loaded into the cache pattern $P_{k\ (k>=1)} \in P$ as a batch, is defined as follows:

$Q_i\ \delta\ Q_j. \Leftrightarrow \{ (\exists) Q_k \in Q /\ h (Q_k, Qi) = h (Q_k, Qj), _{k>=1}\}$

Observation. The equivalence class of query Q_i is defined as a set:

$[Q_i] = \{ Q_j \in Q / h (Q_j, Q_i) = constant, where\ i, j >=1 \}$

In the above example, Q_i, Q_j, Q_k, Q_r, Q_c form one equivalence class say E_1 while Q_p, Q_d, Q_e forms the other equivalence class say E_2. If we replace the pipelined edges with equivalence classes under the relation δ, we obtain the schedule graph of queries. The figure 4 shows the schedule graph with equivalence classes.

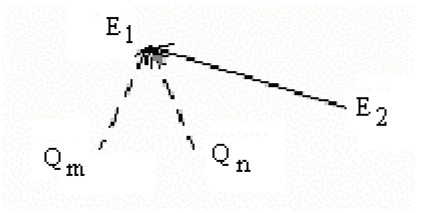

Figure 4. The compressed graph with equivalence.

The Scheduling, Caching and Pipelining problem is reduced to a minimum cost path problem, and the compressed graph contains vertices that are Done or Cached where *Done* is a subset of equivalence classes described above and *Cached* is a subset of nodes in the original uncompressed plan DAG, which are in the subDAGs rooted at the *Done* nodes.

Diwan et al. [1] partition the vertices in the DAG into equivalence classes, such that any two vertices are in the same equivalence class if they are connected by a path of pipelined edges. Let us define α this equivalence relation.

Further we show that the union of all equivalence classes of queries from the cache patterns is the set of cache patterns P.

Proposition 2 Let be a cache pattern (P, Q), where P is a set of cache patterns, $P \neq \phi$, Q is a set of queries loaded into the cache and let α, ρ, δ be equivalence relations on P. Then union of all equivalence classes of queries from the cache patterns is the set of cache patterns P.

Proof:

$\forall\ Q_i \in Q, Q_i$ is a query loaded into the cache, $Q_i \subset P, Q_i$

$\in$ Q is the ith query in the cache pattern P

$Q_i \ \rho \ Q_i \Rightarrow Q_i \in [Q_i] = \{ Q_j \in Q \,/\, \text{CP-MQ}(P_k, Q_i, C_{ik}) = \text{CP-MQ}(P_k, Q_j, C_{js}), \text{ where } P_{k\ (k>=1)} \in P \}$ [1]

$\forall\ Q_j \in Q$, Q_j is a query loaded into the cache, $Q_j \subset P$, $Q_j \in Q$ is the jth query in the cache pattern P

$Q_j \ \delta \ Q_j \Rightarrow Q_j \in [Q_j] = \{ Q_p \in Q \,/\, h(Q_j, Q_p) = \text{constant}, \text{ where } p, j >=1 \}$ [2]

$Q = \{ Q_i \in Q \,/\, Q_i$ is a query loaded into the cache, $Q_i \subset P \}$ [3]

[1], [2] and [3] $\Rightarrow$ Union of all equivalence classes is P

Example: An example of multi-queries optimization is as follows. Let us consider that the query is of type sort-merge-join. We can split the execution of the query into operations which are the equivalence classes that can be cached, and can be reused later.

Let R and S be two relations with the corresponding relational schemas $R(c_1, c_2, \ldots, c_n)$, $S(c'_1, c'_2, \ldots, c'_n)$ and the equi - join of the relations R and S loaded into the cache from the set of cache patterns P:

JOIN (R, S; $R.c_i = S.c'_j$), where $i \in \{1,\ldots, n\}$, $j \in \{1,\ldots, m\}$, $c_i \equiv c'_j$, where p(R) is a predicate with attributes from R and q(s) is a predicate with attributes from S. The query uses the projection (π) on the c_i attribute of the relation R, the JOIN operator ($\bowtie$) and the selection (σ). The execution plan of the sort-merge-join is presented in figure 5:

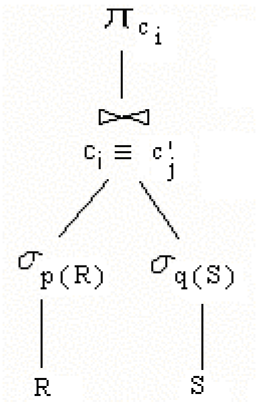

Figure 5. Sort – Join.

Then we can say that sort-merge-join algorithm could be split into two operations, one for selection and sorting the data and one which performs the merge join. This allows the partitions in the equivalence classes to become units that can be cached, and can be reused.

VII. Conclusions

This article presents an approach of the multi-queries optimization with scheduling, caching and pipelining, based on the algebraic pattern. Each cache pattern represents a unique equivalence class in the set of patterns. One query is matched exactly one pattern, whereas each pattern is derived from a set of related queries.

We demonstrated some properties of the equivalence classes defined on the set of cache patterns. We proposed a heuristics of pair queries for query scheduling and caching, that provides a good multi-queries ordering and we defined some equivalence classes of multi-queries from a set of cache patterns. We demonstrated that the union of all equivalence classes of queries from the cache patterns is the set of cache patterns.

References

[1] A.A. Diwan, S. Sudarshan, D. Thomas, "Scheduling and Caching in Multi-Query Optimization", International Conference on Management of Data COMAD, Delhi, India, 2006.

[2] P. Roy, MultiQuery Optimization and Applications, PhD thesis, IIT Bombay, 2000.

[3] S. N. Subramanian, S. Venkataraman, Cost based optimization of decision support queries using transient views, In ACM SIGMOD Intl. Conf. Management of Data, Seattle, WA, 1998.

[4] H. Mistry, P. Roy, S. Sudarshan, K. Ramamritham, "Materialized view selection and maintenance using multi-query optimization", In ACM SIGMOD Intl. Conf. Management of Data, 2001.

[5] J. Shanmugasundaram, G. He, K. Tufte, C. Zhang, D. De-Witt, J. Naughton, "Relational databases for querying XML documents: Limitations and opportunities", In Intl. Conf. Very Large Databases, 1999.

[6] J. Chen, D. J. DeWitt, J. F. Naughton, "Design and evaluation of alternative selection placement strategies in optimizing continuous queries", IEEE Intl. Conf. Data Engineering, 2002.

[7] Sellis, T.K.: Multiple-Query Optimization, ACM Trans. Database Syst. 13(1), pp. 23–52, 1988.

[8] Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S., "Efficient and Extensible Algorithms forMulti Query Optimization", In: SIGMOD,. 2000.

[9] X.Wang, T. Malik, R. Burns, S. Papadomanolakis, A. Ailamaki, "A Workload-Driven Unit of Cache Replacement for Mid-Tier Database Caching",.SIGMOD, 2002.

[10] A. Gupta, S. Sudarshan, S. Viswanathan. "Query scheduling in multiquery optimization", In IDEAS, pp 11– 19, 2001.

[11] N. N. Dalvi, S. K. Sanghai, P. Roy, and S. S. "Pipelining in multiquery optimization", ACM Symp. Principles of Database Systems, 2001.

[12] K.-L. Tan, H. Lu, "Workload scheduling for multiple query processing", Information Processing Letters, 555, pp. 251–257, 1995.

[13] Q.Luo, J. F. Naughton, R. Krishnamurthy, P. Cao, Y. Li, "Active Query Caching for Database Web Servers", ACM SIGMOD Workshop on the Web and Databases, WebDB, 2000.

[14] Q. Luo, J. F Naughton, "Form-Based Proxy Caching for Database-Backed Web Sites", VLDB Conference, Rome 2001.

[15] Sophie Cluet, Olga Kapitskaia, Divesh Srivastava, "Using LDAP Directory Caches", Proc. ACM Principles of Database Systems,1999.

[16] Olga Kapitskaia, Raymond T Ng, Divesh Srivastava, "Evolution and revolutions in LDAP directory caches", Proceedings of the International Conference on Extending Database Technology (EDBT), pp. 202-216, 2000.

[17] Papadomanolakis, S., Ailamaki, A., "AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning", In: SSDBM, 2004.

[18] Nastasescu C., Nita C., Vraciu C., Bazele algebrei, vol I, Editura Academiei Romaniei, Bucuresti, 1986.

[19] G. Graefe, W. J. McKenna, "Extensibility and Search Efficiency in the Volcano Optimizer Generator", In IEEE Intl. Conf. Data Engineering, 1993.

[20] J. Flum, M. Frick, M. Grohe, "Query Evaluation via Tree-Decompositions", Proc. of the 8th International Conference on Database Theory (ICDT'01), volume 1973 of Lecture Notes in Computer Science, pp 22–38, UK, 2001, Springer.