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 rewrites 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
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 = (D_1, D_2, ..., D_n), D_j – range of values, i = 1, ..., n, j = 1, ..., m, where m,n ∈ N*
- a set R of n relations R_1, ..., R_n and each relation consists of a set of attributes:
  R_j(c_{1j}, c_{2j}, ..., c_{mj}, ...), c_{mj} attribute, c_{mj} ∈ D_j, i = 1, ..., n, j = 1, ..., m
  R_j(c_{1j}, c_{2j}, ..., c_{mj}, ...), c_{mj} attribute, c_{mj} ∈ D_j, i = 1, ..., n
- the set Q of all queries in the cache pattern, in which Q_i ∈ 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 attributes loaded into the cache pattern P_k ∈ P and referenced by Q_k ∈ Q, i, j = 1, ..., 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, ..., R_n and an implementation of the JOIN operator defined as follows:

Q_k = JOIN (R_{k1}, R_{k2}, ..., R_{kn}), k_i = 1, ..., n, i >= 1

Let be two queries Q_i, Q_j ∈ Q. Then a subset ρ ⊂ Q x Q is a binary relation between Q and Q_k.

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

i. Q_i ρ Q_j

ii. Q_i ρ Q_j → Q_j ⊂ Q_i

iii. Q_i ρ Q_j and Q_j ρ Q_k → Q_i ρ Q_k,

∀ Q_i, Q_j, Q_k ∈ 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 ρ Q_j ⇔ CP-MQ (P_k, Q_i, C_{ik}) = CP-MQ (P_k, Q_j, C_{jk}), means Q_i and Q_j access the set of attributes, that are loaded into the same cache pattern P_k (k >= 1) ∈ P.

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

[Q_i] = { Q_j ∈ Q / Q_i ρ Q_j }

Definition 5. The equivalence class of query from the cache pattern P_k (k >= 1) is the set:

[Q_i] = { Q_j ∈ Q / CP-MQ (P_k, Q_i, C_{ik}) = CP-MQ (P_k, Q_j, C_{jk}), P_k (k >= 1) ∈ 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 ≠ ∅, 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 ∈ [Q_i], ∀ Q_i ∈ Q set of queries loaded into the cache.

In particular, [Q_i] ≠ ∅

ii. [Q_i] = [Q_j] ⇔ Q_i ρ Q_j where Qi and Qj are related queries

iii. [Q_i] and [Q_j] are equivalence classes [Q_i] = [Q_j] or [Q_i] ∩ [Q_j] = ∅

Proof:

i. Q_i ρ Q_j → Q_i ∈ [Q_i], ∀ Q_i ∈ Q

ii. Let us demonstrate "⇔": let us suppose that [Q_i] = [Q_j]

We have Q_i ∈ [Q_i] then Q_i ∈ [Q_j] → Q_i ρ Q_j

Let us prove "⊂": let us suppose that Q_i ρ Q_j and Q_i and Q_j are related queries

We must demonstrate that [Q_i] ⊂ [Q_j] [1]

Let x be, x ∈ [Q_i] then x ρ Q_j, but ρ is transitive then
Advances in Electrical and Computer Engineering

Volume 10, Number 2, 2010

x ∈ Q3, x ∈ Q3 = [Q3] ⊂ [Q3]

Let us demonstrate that [Q3] ⊂ [Q3]

Let x be, x ∈ [Q3] then x ρ Q3, but ρ is transitive then
x ρ Q3, x ∈ Q3 = [Q3] ⊂ [Q3]

1 and 2, [Q3] = [Q3]

Let us assume that [Q3] ∩ [Q3] ≠ ∅ then x ∈ [Q3]

but ρ is symmetric Q3, ρ x Q3, but Q3 = (from ii) [Q3] = [Q3].

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: Q1 : (R ⨿ S) ⨿ T and
Q2 : (R ⨿ S) ⨿ U. If we evaluate the two queries
separately, then we must recompute R ⨿ S. If we
materialize the result of R ⨿ 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 ⨿ S to both the
queries, we do not have to recompute the result of R ⨿ 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) space can be represented as in
figure 2:

[Diagram of Logical Query DAG space]

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
a 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 ⨿ S to both the
queries, we do not have to recompute the result of R ⨿ 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) space can be represented as in
figure 2:

[Diagram of Logical Query DAG space]

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
a 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 ⨿ S to both the
queries, we do not have to recompute the result of R ⨿ 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.
known that query scheduling and caching problem is NP-complete.

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.

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 $\alpha$ 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, $P \neq \emptyset$, $Q$ is a set of queries loaded into the cache and let $\alpha$, $\rho$, $\delta$ 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:

$$\forall \ Q \in Q, Q_i \text{ is a query loaded into the cache, } Q_i \subset P, Q_i,$$
\[ Q \in Q \] is the \(i\)th query in the cache pattern \(P\) 
\[ Q_i \varrho Q_i \rightarrow Q_i \in \{ Q_i \} = \{ Q_i, \in Q / \text{CP-MQ (P_k, Q, C_k)} = \text{CP-MQ (P_k, Q, C_k)} \}, \text{where } P_k (k \geq 1) \in P \]  \[ \forall Q_j \in Q_i, Q_i \text{ is a query loaded into the cache, } Q_j \subseteq P, Q_j \in Q \text{ is the } j\text{th query in the cache pattern } P \] 
\[ Q_i \varrho Q_j \rightarrow Q_j \in \{ Q_j \} = \{ Q_j \in Q / h (Q_j, p) = \text{constant}, \text{where } p \in P \geq 1 \} \] 
\[ Q = \{ Q_i \in Q / Q_i \text{ is a query loaded into the cache}, Q_i \subseteq P \} \] 

**REFERENCES**


