On Using Materialized Views for Query Execution in Distributed Real-Time Database Management Systems

© Alexander Zharkov

Penza State University
zharkov@sura.ru

Abstract

This paper describes how materialized views could be used in distributed Real-Time Database Management System. We provide algorithm for dynamic materialized views building and cost evaluation. The main difference of provided method is that we take into account temporal properties of base relations and data manipulation operations. Experimental part provides algorithm which builds materialized view for given subset of physical query execution plans.

1 Preface

Real-time database management systems are the database management systems which can provide reliable queries execution in predictable time. Real-time databases use timing constraints to provide valid query results. Inability of conventional databases to work in such circumstances is the main reason to use real-time databases in a wide range of modern industrial applications.

The main are of our interests is the process control applications and using real-time database .management systems to store data for SCADA systems and they components. Modern real-time system should be able to store temporal data, handle time-critical queries, support priority scheduling and interact in dynamic environment. Our previous work was devoted to the scheduling policies in real-time database management systems. In this work we have concentrated on effective queries processing in distributed environment with some restrictions. The restrictions to the testing environment reflect some features of the commonly used computational system in the domain area we are interested in – process control.

Let us review the main features of the computational system which we are using in test environment:

- computational system is heterogeneous – hosts have different workload and different functions. There are 3 type of nodes – “server”, “sensor” and “user”;
- transactions generated by “user” node have are mostly user SQL queries;
- most part of user queries belongs to the limited subset of SQL query templates;
- lifecycle of the system is divided into two parts design-time (when system could be tuned or pre-optimized) and work-time (when optimization tasks should not use mush time);
- user transactions usually take much longer time then sensor or system ones.

These features exert influence on transaction handling and queries processing of the Real-Time Database Management System (RT DBMS). Transaction handling methods for such systems were described in our earlier work [10]. Current work is based on transaction policies handling methods used in [10] and DBMS prototype described in [9]. Transaction handling methods proposed in [10] helps to decrease critical transactions miss ratio but increases user transactions miss ratio. One of the ways to find the balance between user and system transactions is using the distributed database instead of the client-server architecture to provide clients with more information. To implement such approach we have used DBMS engine with materialized views support on client side (“user” nodes). This work describes set-theory model in which our prototype is based on

Related Works

Our research area is placed in the intersection of different areas such as transactions processing, queries optimization, real-time databases, active and temporal database management systems. Common transaction handling aspects in Real-Time DBMS were considered in [1, 2, and 4]. Issues with Timing constraints were described in [3]. Work [5] describes commonly use optimization techniques. We have used some of these techniques in implementation of our query subsystem. Modern line of investigations in multi-query optimizations which used in materialized views support subsystem represented in [6]. Works [7] and [8] are concerned with queries processing which uses both conventional relations and materialized views to produce query result. Our work [9] and [10] describes real-time database management system prototype and transaction handling policies used in our experiments.

♣ **Proceedings of the Spring Young Researcher's Colloquium On Database and Information Systems SYRCoDIS, St.-Petersburg, Russia, 2008**

In materialized views support system we have used elements of the lattice theory [11] to describe some properties of the materialized views support subsystem.

2 Query Execution Plan Representation

The materialized views management subsystem uses the following input information:

- query plans batch;
- source database structure;
- statistical information provided by main server database;

During subsystem initialization materialized views management subsystem transforms input query batch to internal format which helps to minimize execution and update costs. We use lattice representation based query batch as optimal structure for update traces. Internal representation contains mapping from physical query plan items to materialized data items and update support structure.

One of tasks is transformation from physical query plan to internal representation. Usually query execution plan represented by Directed Acyclic Graph (DAG). Formally query execution plan used in our experiments could be represented as follows:

$$G_p = \langle T_{op}, O_p, D_p, R_p \rangle, (1)$$

where T_{op} – set of supported operations;

O_p – operational nodes set;

D_p – data nodes set;

R_p – arcs set which represents relations between nodes;

This model has the following restrictions:

- Operational nodes could be connected only with data nodes and data nodes could be connected only with operational nodes;
- Operational nodes have only one upcoming arc and one ore more incoming arcs (for conventional relation operations no more then 2 arcs)
- Data nodes can have no more then one upcoming and one incoming arc;
- Data nodes can represent not only conventional relations but also other data nodes like indexes or hashes;

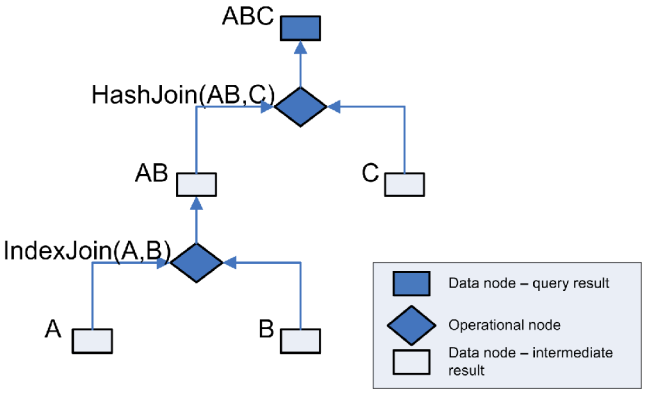

Figure 1— Query Execution plan

Input query plan example represented in figure 1.

Here T_o ={Join, Union, Projection, Selection},

O = {Join(A,B), Join(AB,C)},

D = {A, B, C, AB, ABC},

R = {(A, Join(A,B)), (B, Join(A,B)), (Join(A,B), AB), (AB, Join(AB, C)), (C, Join(AB, C)), (Join(AB, C), ABC)}

A, B, C – source database relations and AB, ABC – results of the join operation.

Physical query execution plan in our subsystem is represented as

$$G_p^{RT} = \langle T_p^{RT}, O_p^{RT}, D_p^{RT}, R_p^{RT} \rangle, (2)$$

where $T_p^{RT} = \{$ScanSelect, ScanSelectID, ScanIndex, ScanSort, NestedLoopsJoin, NestedLoopsIndexJoin, HashJoin$\}$, a set of operational nodes types;

$$D_p^{RT} = \left\{ d = (j, x) : j = \overline{1, l}, l = \left| D_p^{RT} \right|, x \in X_{OBJ}^{RT} \right\}$$

(3) — data nodes set which represents the pair: data node index j and data object x which belongs to X_{OBJ}^{RT} all database objects;

$$O_p^{RT} = \left\{ \begin{array}{l} o = (i, m, A_o) : i = \overline{1, n}, n = \left| O_p^{RT} \right|, \\ m \in T_p^{RT}, \\ A_o = \{ a_o \in A(x_i) \mid x_i \in X_{OBJ}^{RT} \} \end{array} \right\} \quad (4)$$

— operational nodes set. Each node consists of node identifier, transaction type and set of attributes used in transaction execution;

$$R_p^{RT} = \{ r = (o_i, d_j) : o_i \in O_p^{RT}, d_j \in D_p^{RT} \} \cup \{ r = (d_j, o_i) : o_i \in O_p^{RT}, d_j \in D_p^{RT} \}$$

(5) — set of relations between operational and data nodes. These relations are relations of consequence. Set R_p^{RT} represents ordering relationship and the G_p^{RT} model itself is partially ordered on consequence relation.

Query execution batch

Query execution batch is represented as

$$P_{queries} = \langle G_{plans}, Q_{queries}, R_{QP} \rangle, (6)$$

where $Q_{queries}$ — a set of the parameterized queries which are used for quick access to the materialized results;

G_{plans} — set of query execution plans generated for each query. Each query plan is represent as G_p ,(2);

$$R_{QP} = \{ r_{QP} : r_{QP} = (q, G_p), q \in Q_{queries}, G_p \in G_{plans} \}$$

— set of relations which represents mapping between initial query and appropriate subset of query execution plans.

3 Internal materialized views representation

During materialized views management system initialization query execution plans batch is transformed to the internal representation suitable for regular update in real-time mode. For this purpose query plan elements are sorted in and combined into single batch where elements are ordered by inclusion relation to minimize updates count. In our system materialized views batch is represented as lattice which is defined as:

$$L_{\Psi} = \Psi(V^*, R^{\prec}), (6)$$

where V^* – lattice nodes set which corresponds to G_p query execution graphs;

$R^{\prec}$ – model signature which specifies ordering relationship on lattice nodes set.

This model should satisfy to the following restrictions [11]:

- model should have antireflexiveness property– $\forall a \in V^*, \langle a,a \rangle \notin R^{\prec}$ (7);
- model should have asymmetric property – $\forall a,b \in V, \langle a,b \rangle \in R^{\prec} \Rightarrow \langle b,a \rangle \notin R^{\prec}$ (8);
- model should have transitivity property – $\forall a,b,c \in V, (\langle a,b \rangle \in R^{\prec} \text{ and } \langle b,c \rangle \in R^{\prec}) \Rightarrow (\langle a,c \rangle \in R^{\prec})$ (9);

Every node $v^* \in V^*$ represents object which could be materialized. Each lattice node is based on query execution plan node and could be defined as follows:

$$v^* = \langle v', A_V, C_{cost}, X_{data} \rangle, (10)$$

where $v' \in D_P^{RT} \cup O_P^{RT}$ – operational or data node from $G_p \in G_{plans}$ graph nodes set. This dependency between lattice node and query execution plan node is used during query evaluation.

$$A_V = \begin{cases} A_o, v' \in O_P^{RT} \\ \{a_V \mid a_V \in A(x), x \in X_{OBJ}^{RT}\}, v' \in D_P^{RT} \end{cases}$$ – database relation attributes set from the attributes to be materialized. Here A_o – set of attributes used in operational node for cases when this lattice node is based on operational nodes set. $A(x)$ – x relation schema, which is base relation for data node for cases when lattice node is based on data node of the query execution plan;

$C_{cost} = \{C_{reuse}, C_{update}, C_{retrieve}\}$ – set of cost estimates for this materialized view node, where C_{reuse} – cost of materialized view node reuse in cases if node will be materialized, C_{update} – data update cost for cases when node is materialized, $C_{retrieve}$ – data retrieving cost for cases when node is not materialized and appropriate query is executed on main database server. These estimates could be enhanced but it was enough for our prototype;

$$X_{data} = \begin{cases} \varnothing \\ \{x \mid x \in X_{OBJ}^{RT}\} \end{cases}$$ –data relations set created for data materialization in this node. This set is empty for cases when data should not be materialized in current node. If this set is not empty it contains set of data objects which contains base operational nodes result.

Figure 2 represents internal materialized views representation generated by visualisation part of our lattice building test application

4 Query execution plans batch transformation method

This part describes method used for transformation query execution plans batch to the internal materialized views representation. This method is based on building lattice using transitive closure for identifying base nodes.

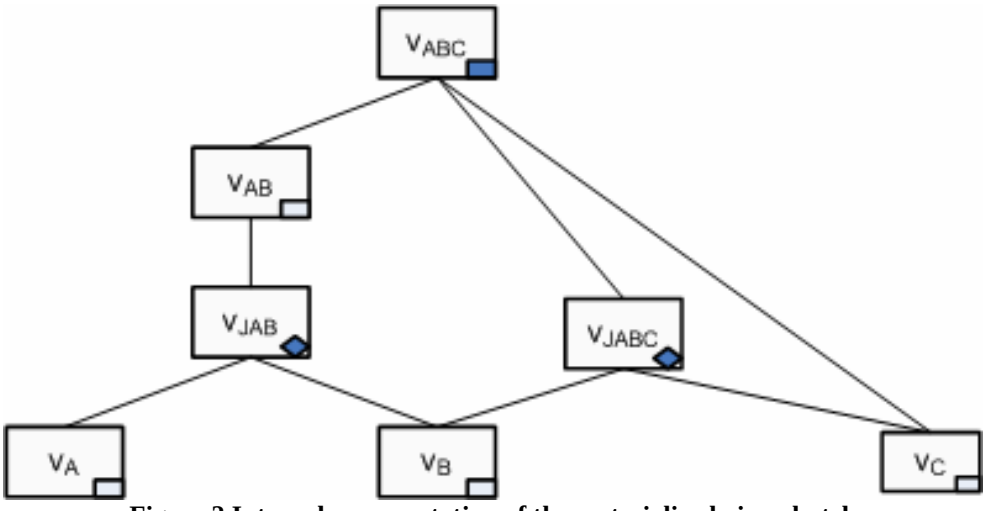

Figure 2 Internal representation of the materialized views batch

Algorithm contains three main parts:

- Building generating sets. Each potential node has corresponding generating set which contains set of attributes used in operational or data node.
- Building lattice which corresponds to rules (7), (8) and (9)
- Cost evaluation for each node and identifying nodes to be materialized.

Generative sets building algorithm

Let V^* materialized view set to be filled. At the first step this set is empty. Let G_{plans} – query execution plans set from $P_{queries}$ then, for each query execution plan $G_P \in G_{plans}$ we should perform the following steps:

1. For each data node $d = (i, x) \in D_P^{RT}$

1.1 Build attributes set $A_d \subseteq A(x), x \in X_{OBJ}^{RT}$, which is subset of the base relation schema.

1.2 Create node $v^* = \langle v', A_V, C_{cost}, X_{data} \rangle$ where $v' = d$, $A_V = A_d$, $X_{data} = \varnothing$

1.3 Add node v^* to the V^* set

2. For each operational node $o = (i, m, A_o) \in O_P^{RT}$

2.1 Create node $v^* = \langle v', A_V, C_{cost}, X_{data} \rangle$, where $v' = o$, $A_V = A_o$, $X_{data} = \varnothing$

2.2 Add node v^* to the V^* set

3. Create node $v^{\cup}$, which has $A_V = \bigcup_{i=1,\overline{|V^*|}} A_V(v_i^*)$, $X_{data} = \varnothing$. This node represents structural unit – superset of all attributes.

4. Add node $v^{\cup}$ to the V^* set

As the result we have V^* set, which is the basis for lattice building.

Lattice signature building algorithm

Second step goal is to build lattice signature $R^{\prec}$ which represents inclusion relation defined on V^* set. This step contains consecutive pair wise intersection for $A_V(v_i^*), v_i^* \in V^*, i = \overline{1,n}, n = |V^*|$ sets. If intersection result is not empty set then new set is used as new node in lattice. And this new node is added as base for two nodes used in intersection.

This algorithm could be divided into two parts:

- Subroutine which adds relation to the relation set and checks new relations to satisfy (7), (8) and (9) restrictions;
- Main iterative part where V^* set elements are looked through.

Let us review add subroutine:

Subroutine Add (A_V , A_V^*)

1. $i = 0$, $n = |R^{\prec}|$

2. If $i < n$, then move to the step 3, else move to the step 8

3. If $r_i = r(x, y) : x \equiv A_V$, then move to the step 4, else move to the step 7

4. If $y \equiv A_V^*$, then move to the step 9, else move to the step 5

5. $z = A_V^* \cap y$

6. If $z \equiv A_V^*$, then move to the step 9, else move to the step 7

7. $i = i + 1$, move to the step 2

8. Add relation $r(A_V, A_V^*)$ to the $R^{\prec}$ signature

9. Exit

Let us review main iterative algorithm

1. $i = 0$, $n = |V^*|$

2. If $i < n$, then move to the step 3, else move to the step 12

3. $j = 0$

4. If $j < n$, then move to the step 5, else move to the step 11

5. $A_V^* = A_{Vi} \cap A_{Vj}$

6. If $A_V^* \neq \varnothing$, then move to the step 7, else move to the step 10

7. If $\exists A_{Vk} \equiv A_V^* : A_{Vk} \in V^*$, then move to the step 8, else move to the step 10

8. Call Add(A_{Vi} , A_{Vk})

9. Call Add(A_{Vj} , A_{Vk})

10. $j = j + 1$, move to the step 4

11. $i = i + 1$, move to the step 2

12. Exit

This algorithm is used to find the most frequently used nodes. But sometimes (for example in cases when there are not so much candidates to the materialization) we can use method less strict method. This enhances base set for cost estimation and could give us more efficient result for small lattice.

The algorithm for building additional nodes

1. $V_{current}^* = V^*$

2. If $|V_{current}^*| > 0$, then move to the step 3, else move to the step 17

3. $i = 0$, $n = |V_{current}^*|$

4. If $i < n$, then move to the step 5, else move to the step 16

5. $j = 0$

6. If $j < n$, then move to the step 7, else move to the step 15

7. $A_V^* = A_{Vi} \cap A_{Vj}$

8. If $A_V^* \neq \varnothing$, then move to the step 9, else move to the step 14

9. If $\exists A_{Vk} \equiv A_V^* : A_{Vk} \in V_{current}^*$, the move to the step 12, else move to the step 10

10. Add A_V^* to the V^* set

11. Add A_V^* to the V_{next}^* set for the next iteration

12. Call Add(A_{Vi}, A_{Vk})

13. Call Add(A_{Vj}, A_{Vk})

14. $j = j + 1$, move to the step 6

15. $i = i + 1$, move to the step 4

16. $V_{current}^* = V_{next}^*$, move to the step 2

17. Exit

This algorithm uses the same subroutine Add as previous algorithm.

Cost estimation for lattice nodes

We use iterative cost estimation method which evaluates cost for each node using optimal cost evaluation for base nodes. The criterion function is defined as $F = \sum_{v_q \in V_{query}} \min\left(Cost'\left(v_q, T_{use}\right)\right)$ (11)

Cost' is subsequent query execution estimation and is defined as:

$$Cost'\left(v, T_{use}\right) = \min\begin{pmatrix} C_{reuse}\left(X_{data}\right) + C_{update}\left(X_{data}, T_{use}\right), X_{data} \neq \varnothing, \\ C_{retrieve}\left(X_{data}, T_{use}\right), X_{data} = \varnothing \end{pmatrix} \quad (12)$$

Retrieve cost is defined as

$$C_{retrieve} = \sum_{v' \in V'} \min\left(Cost\left(v', T_{use}\right)\right), \quad (13)$$

where $V' \subset V^*, V' = \left\{v' : \exists r = r\left(v, v'\right), r \in R^{\prec}\right\}$ – is set of node which are base nodes for current node.

5. Implementation results

Proposed materialized views model was used in distributed real-time database management system prototype for client transactions performance evaluation. For debugging and visual representation of the materialized views special demonstration program was developed which uses algorithms described in section 4 and builds picture represented in figure 2. Algorithms from section 4 were implemented both in real-time DBMS prototype and in test application.

In experimental investigation we plan to focus on 3 main directions:

- experiments with the test application which can visualize internal materialized views representation – this part needed to investigate various combinations of the query batches;
- set of experiments with transaction handling policies to verify what parameters should be used in "user" nodes and "server" nodes;
- set of experiments with different objects to be materialized – the main idea of this part is to investigate how temporary objects (such as temporary indexes and hashes) could be used to increase query processing effectiveness.

For now we have completed first part of experiments – building visual representation of the materialized views based on the user queries (you can see example picture in Figure 2).

6. Further Work

As further work we consider performing experiments on rest two directions using implemented algorithms as part of the distributed database management system prototype. For now we have prepared test database which schema corresponds to the typical database used in process control.

Another direction of the further work is integration of the proposed algorithms to the commercial system which will provide database-like access to the OPC-server data with ability to transfer data to the mainstream database management systems.

References

[1] K. Ramamritham. Time for Real-Time Temporal Databases. Dept. of Computer Sc., Univ. of Massachusetts, 1995

[2] J. R. Haritsa, K. Ramamritham. Real-Time Database Systems in the New Millennium. Dept. of Computer Sc., Univ. of Massachusetts, 1999

[3] Chanjung Park, Seog Park, Sang H. Son. Multiversion Locking Protocol with Freezing for Secure Real-Time Database Systems. IEEE transactions on knowledge and data engineering, vol. 14, no. 5, september/october 2002

[4] John A Stankovic, Marco Spuri, Marco Di Natale, Giorgio Buttazzoy. Implications of Classical Scheduling Results For Real-Time Systems. June 23 1994

[5] Joseph M. Hellerstein. Optimization and Execution Techniques for Queries with Expensive Methods. Doctor of Philosophy dissertation. University of Wisconsin-Madison, 1995

[6] Prasan Roy. Multy-Query Optimization and Applications. Doctor Of Philosophy degree thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, 2000

[7] Hoshi Mistry, Prasan Roy, S. Sudarshan, Krithi Ramamritham. Materialized View Selection and Maintenance Using Multi Query Optimization. ACM SIGMOD Conference Proceedings, 2001

[8] Jiratta Phuboonob, and Raweewan Auepanwiriyakul. Selecting Materialized Views Using Two-Phase Optimization with Multiple View Processing Plan. International Journal of Computer

and Information Science and Engineering. Volume 1, Number 2, 2007

[9] A. V. Zharkov. Distributed Real-Time Database Management System Prototype for Transaction Handling Methods Simulation. In Proceedings of scientific and technical conference "Microsoft Technologies in Programming Theory and Practice". N. Novgorod, 2007

[10] Zharkov A. Performance Evaluation of Transaction Handling Policies on Real-Time DBMS Prototype. Proceedings of the Fourth Spring Young Researcher's Colloquium on Database and Information Systems (SYRCoDIS'2007). Institute for System Programming of the Russian Academy of Sciences. Moscow, 2007. http://CEUR-WS.org/Vol-256/

[11] Birkhoff G. Lattice theory. Providence. Rhode Island. 1967