Performance Evaluation of Transaction Handling Policies on Real-Time DBMS Prototype

© Alexander Zharkov

Penza State University
zharkov@sura.ru

Abstract

We have implemented Soft Real-Time DBMS prototype called ZDBMS. Using this prototype we have investigated performance for commonly used policies EDF, LSF and for new transaction handling policy SPF oriented for using in cases when transactions have different static priorities. The results represent these policies performance for equal sets of experiments.

1 Preface

At present more and more industrial applications interacts in real time. Some of them are manipulating with large amounts of information. So they need in real-time databases and appropriate transaction handling. There is a wide area of industrial applications concerned with process control and SCADA systems which need in specialized real-time database servers. These systems have some distinctive features such as:

- lifecycle is divided into two parts — design-time and work time;
- database clients have limited subset of SQL query templates, and these templates are constant in work-time part;
- sensor and system transactions have different static priorities which reflects subsequent transaction semantics;
- user transactions usually takes much longer time then sensor and system ones;

These features exert influence on transaction handling of Real-Time Database Management System (RT DBMS) used with SCADA system. Traditional works on RT DBMS scheduling policies considers transaction and data consistency taking into account only transaction deadlines and data deadlines. We propose new transaction handling policy which takes into account transaction flow irregularity represented by different static transactions priorities.

Another problem is that some critical (high-priority) sensor or system transactions fails due-to long user query transactions are executed on database server. We decided to use the mechanism of precompiled materialized views for remote database clients. Although influence of materialized views on system performance is not investigated in this paper materialized views management subsystem is the part of client system architecture.

In this paper we describe Soft Real-Time DBMS prototype architecture which implements model for investigating transaction handling policies. In performance evaluation distributed system is used to minimize influence clients (transaction generators) on database server performance.

♣ Proceedings of the Spring Young Researcher's Colloquium On Database and Information Systems SYRCoDIS, Moscow, Russia, 2007

Related Works

There are a lot of works concerned with transaction handling policies so we will mention only papers which results have influenced on this work. Common transaction handling aspects in Real-Time DBMS were considered in [1, 2, 4]. Issues with Timing constraints were described in [3]. Works [5] and [6] are concerns with practical implementation of real-time transaction handling. Real-Time DBMS peculiarity is that research in this area tightly correlated with investigations in area of temporal and active database management systems. Survey on active databases represented in [7] while temporal DBMS is fully described in [8]. During developing query subsystem we investigated commonly used query optimization techniques (represented in [9]). Modern line of investigations in multi-query optimizations which used in materialized views support subsystem represented in [10].

2 Supported transaction model

Supported transaction model is based on composite transactions which can be divided into subset of micro-transactions (atomic transactions such as add row, modify row, etc.). If atomic transaction fails all composite transaction is being revoked.

Transaction taxonomy represented in Figure 1. It describes transactions RT transactions which can appear in typical SCADA system and respectively in appropriate Real-Time DBMS.

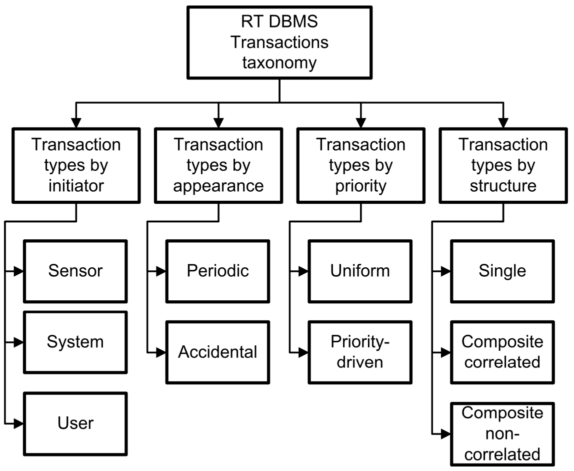
RT DBMS Transactions taxonomy
Transaction types by initiator
Transaction types by appearance
Transaction types by priority
Transaction types by structure
Sensor
System
User
Periodic
Accidental
Uniform
Priority-driven
Single
Composite correlated
Composite non-correlated

Figure 1. RT DBMS Transactions taxonomy

Formally each atomic transaction can be represented as follows:

$$w_{micro} = \langle x, d, p, m, t_a, t_{rvi}, t_{exec}, t_{end} \rangle, \quad (1)$$

where x — db object which corresponds to given transaction;

d — data associated with atomic transaction;

p — static transaction priority;

m — atomic transaction type, $m \in M$, M={Add, GroupAdd, Insert, GroupInsert, Modify, GroupModify, ScanModify, Delete, GroupDelete, ScanDelete, ScanSelect, ScanSelectID, ScanIndex, ScanSort, NestedLoopsJoin, NestedLoopsIndexJoin, HashJoin} — set of possible atomic transaction types;

t_a — timestamp represents arrival time;

t_{rvi} — relative validity interval — time interval when data associated with transaction keeps its validity;

t_{exec} — timestamp represents execution beginning;

t_{end} — timestamp represents transaction end time (failure or success);

3 Real-Time DBMS prototype architecture

Implemented experiment system contains two types of RT DBMS prototype applications:

- dedicated server — RT DBMS server which handles clients requests
- transaction generator — this application emulates clients activity and combines pure transaction generator with RT DBMS server to handle queries to materialized views (materialized views influence is not evaluated in this paper).

Both application types are based on common system architecture represented in illustration below (Figure 2). Main components are: Network Manager, Transaction Decomposer, Query Manager, Priority Manager, Transaction Scheduler, Transaction Pool and DB-objects Manager

Network Manager

Network Manager handles all communication between current server and over applications. Communication is based on own protocol, which supports control packets and transaction packets. This protocol is datagram protocol based upon UDP protocol. Network Manager stores its own settings (clients/servers map, datagram resend policy etc) in XML configuration file which can be loaded from the disk or sent throw the LAN by using special control packet.

Transaction Decomposer

Transaction decomposer receives waiting transaction from the Network Manager queue, decomposes transaction into atomic transactions and passes micro-transactions subset either to Query Manager (for user queries) or to Transaction Scheduler queue.

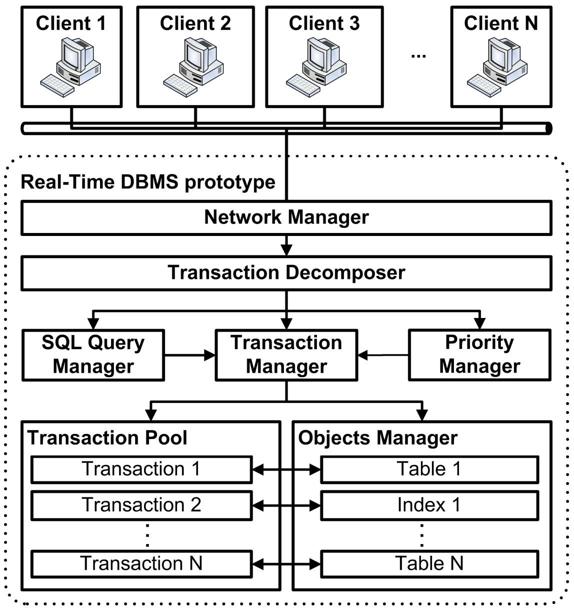

Figure 2. Real-Time DBMS prototype architecture

Query Manager

Query manager handles composite correlated transactions which represents compiled SQL queries. The main purpose of query manager is to reconstruct logical query plan represented as DAG (Directed Acyclic Graph) and to transform it to appropriate physical DAG based on db-objects or materialized views.

Priority Manager

Priority Manager is a class which implements all necessary methods for building dynamic transaction priority. By changing special tuning parameters it can handle priorities using one of scheduling policies.

Transaction Scheduler

Transaction scheduler is component which handles all atomic transactions sets and assigns them into unoccupied threads from transaction thread pool. Transaction Scheduler has special thread used for synchronization — all scheduling actions are performed within this thread to avoid resource management deadlocks.

All atomic transaction sets are waiting for their assignment in queue. In each scheduling step Transaction Scheduler checks this queue for failed transactions to return failed transaction back to the client.

When assigning transaction to the thread Transaction scheduler uses priority returned by Priority Manager in compliance with current transaction handling policy.

Transaction Pool

Transaction Pool is a set of threads capable of transaction executing. When transaction is assigned to the subsequent thread this thread locks object need for the first atomic transaction in the set. For locking Db object two-phase locking method used. First Lock is made by synchronization thread of Transaction Scheduler and second phase is performed by transaction executing thread. In execution step transaction is checked for data consistency and is aborted if deadline or data-deadline achieved. So all the transaction scheduling policies has data consistency check, but – DC suffix is omitted.

DB-objects Manager

Objects Manager is the component which represents In-Memory database. It supports the following object types: tables; indexes (B-tree index); hashes; materialized views;

All objects represented in block-list (or paged) structure. Access to each object can be locked on page-level. Each object has a set of temporal attributes used in priority handling. Let X is object in RT DBMS then it has the following attributes:

LUT(X) — last update time of the X object — this time is calculated automatically by system;

RVI(X) — relative validity interval — this value is set to each object at the design time. It represents time interval in which object keeps its validity;

$AVI(X)$ — Absolute Validity Interval is the time interval which can be calculated as follows:

$$AVI(X) = [LUT(X), LUT(X) + RVI(X)] \quad (2)$$

4 Transaction handling policies

All transaction handling (scheduling) policies are based on transaction model represented in (1), temporal object properties LUT(X), RVI(X), AVI(X) and a subset of dynamic transaction properties described below.

Dynamic transaction properties

$DL(T)$ — transaction deadline is defined as follows:

$$DL(T) = t_a + \min(RVI(d), t_{rvi}), \quad (3)$$

$DDL(T)$ —data deadline depends on temporal object properties:

$$DDL(T) = AVI(x) = LUT(x) + RVI(x) \quad (4)$$

$EET(T) = f(x,d,m)$ — estimated execution time is the function which depends on transaction type, transaction data and object statistics.

$ETT(T)$ — execution transaction time is defined as follows::

$$ETT(T) = now() - t_{exec}, \quad (5)$$

where $now()$ — special temporal function which returns current system time.

$SF(T)$ — slack factor is a parameter which takes into account estimated amount of time to transaction deadline. Slack factor is defined as follows:

$$SF(T) = DL(T) - (now() + EET(T)). \quad (6)$$

$PR(T)$ — proposed function which combines traditional priority handling policies (EDF, EDDF, LSF). We define this function as :

$$PR(T) = PR_{EDF}(T) \times (1-\mu) + \frac{\mu}{SF(T) - 1}, \quad (7)$$

where $PR_{EDF}(T)$ (8) — priority handling function which combines EDF and EDDF policies is defined as:

$$PR_{EDF}(T) = (\alpha \times DDL(T) + (1-\alpha) \times DL(T)) \quad (8)$$

Parameters α and μ are tuning parameters and used to handle which policy should be used at current moment.

To include semantic priority to this policy combining functions we introduce priority comparison function PR_c defined as:

$$\begin{aligned} PR_c(T_1, T_2) = \beta \times SIGN(p(T_1) - p(T_2)) + \\ (1-\beta) \times SIGN(PR(T_1) - PR(T_2)) \end{aligned}, \quad (9)$$

where $\beta \in [0,1]$ — tuning parameter which helps to combine static priority $p(T)$ with dynamic priority $PR(T)$ defined in (7);

$$SIGN(v) = \begin{cases} 1, \text{if } v >= 0 \\ 0, \mathit{if}\ v < 0 \end{cases} \text{ — sign function.}$$

As result we have a set of functions which can be used to combine different transaction handling policies at the same time by using tuning parameters α, β, μ.

Transaction handling policies supported by RT DBMS prototype

EDF — Earliest Deadline First — this scheduling policy is historically first. It takes into account only transaction deadlines. Tuning parameters for it according to (7-9) are $\alpha = 0, \beta < 0.5, \mu = 0$.

EDDF — Earliest Data Deadline First — transactions with earliest data deadline served first. This strategy does not takes into account transaction deadline. Tuning parameters for it are $\alpha = 1, \beta < 0.5, \mu = 0$.

Hybrid — Hybrid approach — takes into account both transaction and data deadline. Here tuning parameter α is defined as:

$$\alpha = \begin{cases} \frac{ETT(T)}{EET(T)}, ETT(T) \le EET(T) \\ 1, ETT(T) > EET(T) \end{cases} \quad (10).$$

Other tuning parameters are $\beta < 0.5, \mu = 0$.

HH — Half-Half approach — another way to compromise between transaction and data deadline. Here α is defined as:

$$\alpha = \begin{cases} \frac{ETT(T)}{EET(T)}, \frac{ETT(T)}{EET(T)} \le 0.5 \\ 1, \frac{ETT(T)}{EET(T)} > 0.5 \end{cases} \quad (11)$$

LSF — Least Slack First — highest priority is assigned to transactions with least positive slack factor. Tuning parameters are $\beta < 0.5, \mu = 1$, parameter α does not influence on transation priority for this policy.

EDF-DC and LSF-DC — policies are similar to EDF and LSF except in each execution step they have data consistency check (fails if data deadline detected).

SPF — Static Priority First — newly proposed strategy which takes into account transaction semantics through its static priority. In this strategy static priority is more valuable then dynamic temporal priorities. Tuning parameters for this policy is $\beta > 0.5$, parameters α and μ, depends on secondary temporal policy. This policy could be used in systems which have a variety of transaction types with different semantic priorities.

5 Performance evaluation results

Performance evaluation in this paper is devoted to comparison of proposed SPF policy with EDF and LSF policies. All policies are used with data consistency check, so we omit –DC suffix.

Experiment system has two PC connected through 100 Mbit Ethernet. One PC was used for dedicated server and another one for transaction generator. Dedicated server had test database with 20 tables (each table with 10000 rows, RVI(X)=100ms). Server had transaction pool with 10 threads. Transaction generator had 10 generation threads each generates 2 transactions (with static priority 100 and 500) per time. Generation period varies from 10 to 300 ms. Generated transactions are uniformly distributed within generation period. Relative validity interval for transaction data is t_{rvi}=40ms. Each generated transaction contains 1000 atomic transactions of ModifyRow type.

For each generation period there was 10 test with 1000 transactions generated. After each test application was reloaded. For each test average miss ratio was calculated (miss ration = failed count / total count) and average miss ratio for high-priority (500) and low-priority (100) transactions. Tables 1 and 2 contain performance evaluation results for each policy represented with total miss ratio (Table 1) and miss ratio for high-priority transactions (Table 2). Figure 3 contains appropriate diagram.

From the results we can see that total SPF miss ratio is higher then total miss ratio for EDF and LSF, but miss ratio for high-priority transactions is much better in comparison with EDF and LSF. So SPF policy can be used to minimize high-priority transactions miss-ratio. To decrease low-priority transactions miss ratio in distributed environment client-side materialized views could be used. This approach was approved in SQL manager implementation (with earlier SPF policy version [12]) for SCADA KRUG-2000 DB Server [11].

Table 1. Total Miss Ratio

Period, ms	SPF	EDF	LSF
10	0.961	0.906	0.913
20	0.853	0.823	0.936
30	0.806	0.746	0.7
40	0.608	0.69	0.61
50	0.575	0.619	0.499
60	0.369	0.576	0.409
70	0.337	0.543373	0.344
80	0.32	0.315	0.293
90	0.314	0.262	0.219
100	0.232	0.259	0.215
110	0.178	0.243	0.181
120	0.199	0.208	0.236
130	0.152	0.13	0.143
140	0.109	0.091	0.167
150	0.119	0.189	0.144
160	0.145	0.071	0.125
170	0.098	0.058	0.059
180	0.053	0.05	0.124
190	0.047	0.15	0.104
200	0.033	0.129	0.036

Table 2. Miss Ratio of High-Priority Transactions

Period, ms	SPF-High	EDF-High	LSF-High
10	0.922	0.896	0.902
20	0.706	0.82	0.912
30	0.612	0.742	0.696
40	0.22	0.686	0.604
50	0.17	0.616	0.5
60	0	0.564	0.402
70	0	0.518072	0.336
80	0.006	0.296	0.282
90	0.01	0.248	0.208
100	0	0.254	0.21
110	0	0.222	0.16
120	0.002	0.202	0.214
130	0	0.11	0.116
140	0	0.084	0.146
150	0	0.17	0.11
160	0	0.068	0.114
170	0	0.054	0.052
180	0	0.046	0.092
190	0	0.14	0.06
200	0	0.102	0.034

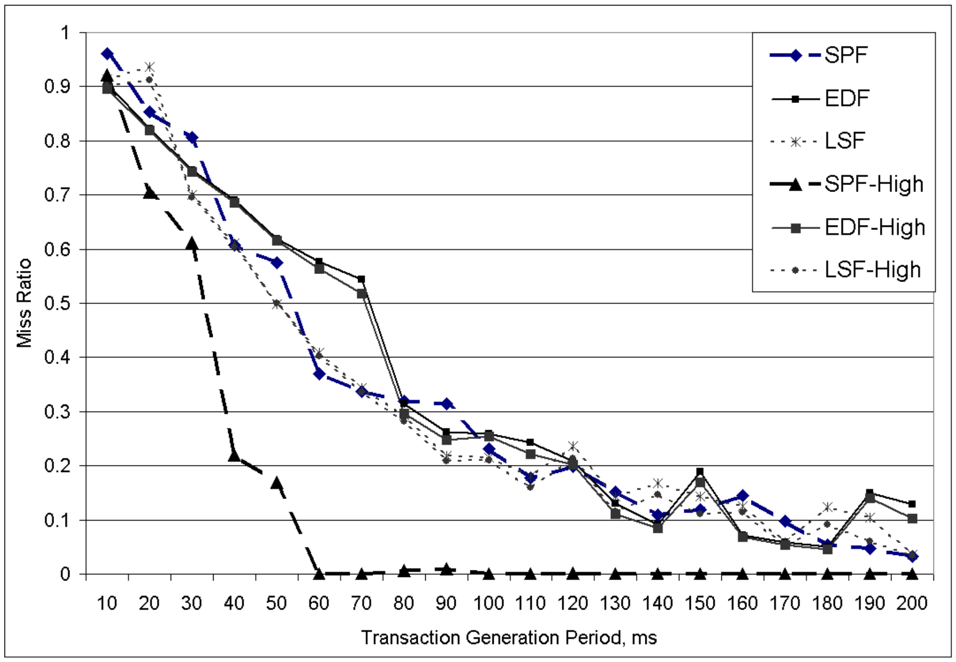

Figure 3. Results diagram for SPF, EDF and LSF performance evaluation

6 Further work

As further work we consider more experiments with scheduling policies with evaluating SPF policy performance in dependence of β tuning parameters. Another direction of further experiments is investigating of materialized views using influence on common system performance. This work is concerned with tuning parameters and scheduling policy for client-side server, which handles materialized views. And as final part we propose the experiments on computational system which simulates real SCADA system with appropriate database structure and hosts configuration.

References

[1] Patrick E. O'Neil, K. Ramamritham, Calton Pu. Towards Predictable Transaction Execution in Real-Time Database Systems. Dept. of Computer Sc., Univ. of Massachusetts, 1995

[2] Shuoqi Li, Ying Lin, Sang H. Son, John A. Stankovic, Yuan Wei. Event Detection Services Using Data Service Middleware in Distributed Sensor Networks. Department of Computer Science, University of Virginia. Kluwer Academic Publishers. Printed in the Netherlands, 2003

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

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

[5] 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

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

[7] Norman W. Paton, Oscar Di´az, Active Database Systems. // ACM Computing Surveys, Vol. 31, No. 1, March 1999, p 63-103

[8] Christian S. Jensen. Temporal Database Management. Dr. techn. Thesis, defended April 2000, http://www.cs.auc.dk/~csj/Thesis/

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

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

[11] A. V. Zharkov. Using SQL to Access Data of SCADA KRUG-2000. In Proceedings of international scientific and technical conference "Automation and Management Problems in Engineering Systems". Penza, 2004.

[12] B. D. Shashkov, A. V. Zharkov. Microtransaction Handling in Real-Time Database Management Systems. In Proceedings of Computer-Based Conference "Contemporary information technologies - 2005". Penza, 2005

[13] 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