The V4DB project – support platform for testing the algorithms used in real-time databases

VÁCLAV KRÓL 1, JAN POKORNÝ 2, JINDŘICH ČERNOHORSKÝ 2

1 Institute of Information Technologies
Silesian University in Opava, Faculty of Business Administration in Karvina
Univerzitní náměstí 1934/3, 733 40 Karviná
CZECH REPUBLIC

2 Department of Measurement and Control
VŠB Technical University Ostrava, Faculty of Electrical Engineering and Informatics
17. listopadu 15, 708 33 Ostrava
CZECH REPUBLIC

Abstract: - Many applications supported by a database management system are characterized by the requirement of timely access to the underlying database today. It is very difficult to achieve guaranteed real time database services when putting a database into a real-time environment because various components can compete for system resources. Previous research in real-time databases has focused primarily on evolution and evaluation of transaction processing algorithms, priority assignment strategies and concurrency control techniques. But for the most part the research efforts are based only on simulation studies with many parameters defined. Our objective is to design and implement an experimental real-time database system suitable for study of real time transaction processing. The experimental system is implemented as an integrated set of the most important functional parts of a veritable real-time database system. It serves as a support platform for performance evaluation of known and new algorithms of the particular processing components, including CPU scheduling, concurrency control and conflict resolution strategies. Because of the strong interactions among the processed components, proposed system can help us to understand their effect on system performance and to identify the most influencing factors.

Key-Words: - Real-time database, transaction processing, CPU scheduling, concurrency control

1 Introduction

The field of real-time database (RTDB) systems is bringing together concepts from two research fields: traditional database management systems and real-time systems. While tasks are considered as the basic schedulable unit in real-time systems, transactions are the schedulable units in database management systems. Transactions in a RTDB system travel through various components until their termination. In contrast to a conventional database management systems where the goal usually is to minimize transaction response times, a real-time database system may be evaluated based on how often transactions miss their deadlines, the average lateness of transactions or the cost incurred in transactions missing their deadlines. RTDB model is presented on fig. 1 to outline the involved resources and to indicate the particular components of real-time transaction processing. The model is adopted from [3].

There are essentially three problems with which RTDB systems must deal: resolving resource contention, resolving data contention, and enforcing timing constraints [5]. So one of the basic concepts of RTDBS consists in integration of all these inevitable aspects.

![RTDB model](image-url)
2 Motivation

Up to now the major part of RTDB research was focused on evolution and evaluation of transaction processing algorithms, priority assignment strategies and concurrency control techniques. Evaluation was usually based on simulation studies except a few exceptions ([4]). Simulations often consist of a number of parameters. The parameters specify maximal count of data items, average count of one transaction data pages, processor time needed to manipulate data items, average disk access time, probability of read vs. write transaction, etc. There is even a study where all the functional blocks are designed as object-oriented and described by means of classes with a number of attributes ([6]). Much less attention was paid to architecture aspects of the operating systems, developed especially for real-time systems and for better support of time critical operations. So two basic drawbacks of the presented research can be defined:

1. For the most part there is only one functional part considered for investigation without any interaction with other system parts. Because of the strong interactions among the various processing components in RTDBS, an integrated approach is necessary.

2. Research work at real-time transaction processing is based on simulation studies only. It is necessary to investigate the real-time transaction processing algorithms in their natural environment to achieve really relevant results. It means that the operating platform for RTDBS is a real-time operating system and the particular RTDBS functional blocks communicate with each other by means of this operating system.

2.1 Primary idea

The main goal of the project is to design and implement an experimental real-time database system suitable for study of real-time transaction processing. The experimental system must be implemented as an integrated set of the most important functional parts of a veritable real-time database system. It would enable testing and performance evaluation of known and new algorithms of the particular functional parts to understand the effect of various processing components on system performance and to identify the most important influencing factors. This is the key point of the proposed experimental system V4DB. The idea is outlined on fig. 2.

2.2 Integrated approach to system construction

One of the most important aspects of our research is to consider the intended experimental real-time database system as one whole composed of integrated particular components. An integrated approach is necessary because even a single component in the system which ignores timing issues can undermine the best efforts of algorithms which do account for timing constraints.

2.3 Implementation in RTOS environment

Without adequate support from the underlying subsystems, none of the scheduling algorithms can guarantee predictable transaction performance. RTDBS building blocks must be integrated with the real-time operating system kernel and other run-time environment building blocks in order to avoid wasteful duplication and provide predictable services. V4DB system is built upon a real-time operating system platform VxWorks to provide reasonable results.

3 System design

The system is currently implemented upon the real time operating system platform VxWorks as a centralized system with memory resident database. Overall design is presented on fig. 3. Oval blocks represent parallel processes while the square blocks are single functional blocks within processes. Some of the system parts contain grayed blocks. The blocks illustrate the possibility of functionality change of the parts. Their runtime behaviour can be changed according to predefined setting before the system start. Particular components are described in more detail in the next chapters.

Physical structure and logical model of the database are the key points of the functional system design. Therefore the database schema is described at first.
3.1. Database

Regarding the project objective as an experimental system and application categories where RTDBS are used to advantage a simple schema is adopted in the following form: Database is divided into predefined count of memory areas. Each area represents some table (or, generally said, some “object”) and consists of predefined count of records (or object “properties”). Records are of the same length for one memory area table just for simplicity. Physical database schema is outlined on fig. 4.

The database schema can be defined by the notation: 
\[ \text{tab name} \mid \text{rec count} \mid \text{rec byte length} \]

- means that there exists a table named Tab01 which has 100 records each of 50 bytes in length.

3.1.1. Database granularity

Since it is very interesting to find the impact of database granularity on the system performance, the granularity can be defined for each table mentioned above. Each table has an additional parameter “granularity”. The parameter stands for the count of logical areas into which each table is divided for the needs of concurrency control during transaction processing. The parameter must fulfill the rules:

\[ 1 \leq g \leq rc \quad \text{and} \quad rc \mod g = 0 \]

- record count in table

The granularity is defined separately for each table, so the parameter can be added to the table definition and the final database schema looks like that:

\[ \text{tab name} \mid \text{rec count} \mid \text{rec byte length} \mid \text{granularity} \]

- means that there exists a table named Tab01 which has 100 records each of 50 bytes in length, the table is divided into two logical areas according to granularity 2.

3.1.2. Logical model

Logical database access just copies the physical database schema. Access to the third record in table Tab01 can be simply defined like Tab01:3. Since there are quite designedly not specified any meanings of the records (in relational terms, the “record attributes”) there is no need to specify anything else for finding the required record.

3.2. Description of the transactions

Transactions are generated by the internal generator. To study the database transaction processing, it must be able to generate transactions which properties are known and set in advance. Therefore it is important to determine all needed parameters and characteristics of the transactions and to describe these parameters suitable enough. We would like to capture the data via the socket interface from the system outside in the future because the internal generator needs some system resources and has certain impact on overall database management system kernel performance.
Fig. 5. Description of transactions

As mentioned above, logical database access results from physical database design and that is the way by which it is defined the access to a particular database tuple. Access to second record of table Tab01 can be written as Tab01:2, etc. Next the four basic database access methods must be distinguished.

<table>
<thead>
<tr>
<th>DB operation</th>
<th>shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>S</td>
</tr>
<tr>
<td>Update</td>
<td>U</td>
</tr>
<tr>
<td>Insert</td>
<td>I</td>
</tr>
<tr>
<td>Delete</td>
<td>D</td>
</tr>
</tbody>
</table>

Table 1. Shortcuts of DB operations

For example, to select Tab01:2, it can be simply written as S/Tab01:2.

Besides these basic principles it is important to describe and work with some other, non-necessarily required parameters that further specify the transaction.

<table>
<thead>
<tr>
<th>DB operation</th>
<th>shortcut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline</td>
<td>T</td>
</tr>
<tr>
<td>Period</td>
<td>P</td>
</tr>
<tr>
<td>Criticality</td>
<td>C</td>
</tr>
</tbody>
</table>

Table 2. Shortcuts of transaction’s RT characteristics

Deadline is the most important parameter of course. We plan adding some other parameters in the future.

3.3. Predispatching

After the admission the transactions are predispatched. Predispatching includes admission control to avoid system overloading and creating the transaction infrastructure. The structure fully describes the transaction definition and all its parameters. All the system functional blocks access this structure, read the transaction setting and write the runtime system messages into the structure. The structure access is protected by binary semaphore. The structure looks like:

```c
typedef struct {
    long int trxId;
    long int timeTickDispatch;
    long int timeTickStart;
    long int timeTickFinish;
    long int deadline;
    long int criticality;
    long int priority;
    int algPriority;
    SEM_ID semTrxInfoStructGuard;
    SEM_ID semTrxProcess;
    char trx_Definition[MAX_LEN];
} TRX_INFO;
```

3.4. Dispatching

In the next step the transaction parameters are extracted and dispatched for execution according the priority assignment policy and the way of transaction processing. There are many ways how the transactions can be processed: each of them can be a single process, they can also be handled by several predefined processes in process pool or by two hit-miss queues ([2]) and so on. So it would be very difficult to cover all the functionality into one block and that is why a specific dispatcher must be created for every used policy (specific version of dispatcher is illustrated by grayed blocks on fig. 3).

Dispatcher has a simple interface. Currently there are two types of dispatcher implemented. The first one assigns process for every new transaction. The second dispatcher distributes the transactions into the predefined processes within process pool.

In any case the priority assigned to a transaction execution process is mapped to a real operating system process priority and the context (transaction) switching is relied on an underlying operating system. This is one of the most important experimental system aspects.

3.5. Processing

When the transaction is scheduled for execution, first it is parsed into particular commands and then the commands are processed by the command executor. System commands are executed immediately while database access must be synchronized through concurrency control. DBQuery block executes the commands on logical level while the resource manager and memory manager work with physical data structures that are described by data dictionary.

3.5.1. Parser

Parser is responsible for parsing the whole transaction definition into the system and db commands. DB commands are described above. The Gap command,
noted as G, is an example of the system command. This command stands for a delay of the execution of a transaction. Such system commands can help us to better simulate the transaction input behaviour.

3.5.2. Command executor
Command executor realizes the parsed commands and pass them further. The execution is strictly sequential in the scope of the transaction. The executor realizes system and DB commands.

3.5.3. DBQuery
This block executes DB commands on logical level. It defines three basic logical DB operations: insertRec, updateRec, deleteRec

3.5.4. Resource manager
Resource manager makes use of the structures defined in the data dictionary and the functions defined in the memory manager. By means of the information from the data dictionary it can access the underlying data storage.

3.5.5. Data dictionary
Physical data structures are described by data dictionary. The dictionary is composed of two parts:
- structure of pointers to the physical database objects
- control structures needed for transaction processing
While in the structure of pointers to database objects is the current snapshot of real data stored, control structures are used for transaction management, in particular for concurrency control. Because of fundamental differences between protocols of concurrency control there must be separate functional blocks implemented for particular concurrency control types as it is in the case of the dispatcher.

3.5.6. Concurrency control
DB transactions compete for access to the data stored in the database. To obtain reasonable performance, multiple transactions must be able to access data concurrently. So before a transaction performs an operation on a data object, it must be processed by concurrency control component in order to achieve the required synchronization. If the transaction’s request for an item is denied, the transaction waits. The waiting transaction is reactivated when the requested data block becomes available.

An implementation of two phase locking protocol, so called 2PL, is under implementation now. It is the most used protocol which belongs to a family of pessimistic protocols. We plan implementing some other kinds of concurrency control protocols in the future, first of all an optimistic protocol to make a basic comparison. However, it turned out that the implementation of concurrency control component requires certain modifications of other system components. A change of concurrency control policy must not have any influence on other system parts from implementation point of view. So there must exist some uniform interface and each of the concurrency control mechanisms must implement the interface to stay quite independent. Fig. 6 illustrates the way of V4DB concurrency control implementation.

There are defined several interface methods that are called at some points during the transaction processing, before-after transaction starts, before-after access methods, etc. The implementation of these methods is relied on concurrency control policy. Besides these methods each of the process performing transaction processing has its own so called continuation semaphore. Concurrency control can take or give the semaphore according to current state and policy used. The semaphore is always polled on a transaction processing level before performing a database action (read, write, delete). Process state switching is performed by the underlying RTOS so the concurrency control is integrated with the real-time operating system kernel in this manner and this essential requirement is fulfilled.

![Concurrency control independence](image)

**Fig.6.Concurrency control independence**
4 CONCLUSION
Currently we are in a final phase of V4DB core database services implementation. The first experimental results will be presented immediately after finishing. It is intended to create the system in such a way that it would be possible to extend it and continue its development later. Future directions can be suggested right now: variable database granularity, indexing algorithms ([12]), backup possibilities or database movement to a disk media.
It is assumed that the created system will be used as the experimental rudiment for future research and for future advancement to study another, more complex properties of RT databases.

Acknowledgement
The support for this research work has been provided by the project 102/06/1742: Experimental real-time database testing system and the project 102/05/H525: The Postgraduate Study Rationalization at Faculty of Electrical Engineering and Computer Science VSB-TU Ostrava, both provided by the Czech Science Foundation.

References: