Timely ACID Transactions in DBMS

Marco Vieira
ISEC-CISUC, Coimbra, Portugal
mvieira@isec.pt

António Casimiro Costa
FCUL, Lisboa, Portugal
casim@di.fc.ul.pt

Henrique Madeira
DEI-CISUC, Coimbra, Portugal
henrique@dei.uc.pt

1. Introduction

Developing database applications with timeliness requirements is a difficult problem. During the execution of transactions, database applications (with timeliness requirements) have to deal with the possible occurrence of timing failures, when the operations specified in the transaction do not complete within the expected deadlines.

In spite of the importance of timeliness requirements in database applications, the transaction engines of database management systems (DBMS) do not assure any temporal property, not even the detection of the cases when the transaction takes longer than the expected/desired time.

Our goal is to investigate ways to add timeliness properties to the typical ACID (Atomicity, Consistency, Isolation, and Durability) properties supported by most DBMS.

2. Existing DBMS transactions

A database is a collection of data describing the activities of one or more related organizations [1]. The software designed to assist in maintaining and using databases is called database management system, or DBMS. The need for such systems, as well as their use, has grown rapidly in the last two decades. A DBMS allows users to define the data to be stored in terms of a data model, which is a collection of high-level metadata that hide many low-level storage details. Most DBMS today are based on the relational data model, which was proposed by E. F. Codd in 1970 [2]. The relational data model is very simple and elegant, and defines a database as a collection of one or more relations, where each relation is a table with rows and columns.

In practice, a typical database application (e.g., banking, insurance companies, all sort of traveling businesses, telecommunications, wholesale retail, complex manufacturing processes, etc, etc) is a client-server system (either a traditional client-server or a three tier system) where a number of users are connected to a database server via a terminal or a desktop computer. The user actions are translated into SQL commands (the relational language used by DBMS) by the client application and sent to the database server. The results are sent back to the client to be displayed in the adequate format by the client application.

A very important notion in DBMS is the concept of transaction [3]. In a simplified view, a transaction is a set of commands that perform a given action and takes the database from a consistent state to another consistent state. Transaction management is an important functionality of DBMS and it is directly related to dependability, particularly in what concerns to concurrency control (essential to assure data integrity) and recovery. Concurrency control is the activity of coordinating the actions of processes that operate in parallel and access shared data, and therefore potentially interferes with each other. Recovery assures that faults do not corrupt persistent data stored in the database tables.

In order to correctly deal with concurrency control and recovery, DBMS transactions must fulfill the following properties (also known as ACID properties): **atomicity** (either all actions in the transaction are executed or none are), **consistency** (the execution of transactions results in consistent database states), **isolation** (the effects of a transaction must be understood without considering other concurrently executing transactions), and **durability** (the effects of a transaction that has been successfully completed must persist, even when the system has a failure after transaction finishing).

3. Timely ACID Transactions

In many situations timeliness is more important than correctness, which means that (approximate) correctness can be traded for timeliness [4]. Similarly, atomicity issues may be relaxed. For instance, in many database applications, when a transaction is submitted and it does not complete before a specified deadline that transaction becomes irrelevant. This means that, the DBMS can automatically rollback the execution of the transaction if the deadline is exceeded. However, as there are no mechanisms implemented in existing DBMS able to detect this kind of timing failures, the automatic abortion of transactions is not possible.

There are also situations, where the database user must be informed when the execution of a transaction do not complete in a specified deadline. For instance, the collection of information about timing failures and the temporal execution of transactions can be used to feed a monitoring component or to tune specific application parameters in order to adapt its behavior to the actual load conditions of the transactional system.

Our proposal is to add the timeliness property to the transactions. As mentioned before, most DBMS support transactions with ACID properties. Our goal is to extend DBMS in order to support transactions with TACID properties (timeliness, atomicity, consistency, isolation, and durability).

4. Timely Computing

One of our objectives is to use the Timely Computing Base (TCB) model [5] as the framework for the work presented in this paper. Essentially, a system with a Timely Computing Base is divided into two well-defined parts: a *payload* and a *control* part. The generic or *payload* part prefigures what is normally 'the system' in homogeneous architectures, and is where applications such as DBMS execute. The *control* part, which we call the TCB, is a comparably much smaller part of the system, which can thus be designed to provide "better" timeliness properties than the payload part of the system. In fact, the TCB must be designed as a synchronous component, whereas the payload part of the system may have any degree of synchronism.

A TCB can be seen as an oracle providing time-related services to applications or middleware components. Therefore, a set of minimal services has to be defined, as well as a payload-to-TCB interface.

Previous TCB implementations [6] were designed to provide a set of generic services, potentially useful to a large range of applications (e.g. multimedia, control, security): a **Duration Measurement** service, a **Timing Failure Detection** (TFD) service and a **Timely Execution** service. Now, when considering the specific case of DBMS with TACID properties, it is important to understand which are the new challenges raised by this kind of application and how can the TCB model be applied in this environment.

As a result of our initial investigations, we identified the following challenges:

Definition of TCB services: in order to optimize and simplify the TCB oracle, it is fundamental to provide just the strictly necessary functionality, which is yet unknown.

Definition of payload-to-TCB interfaces: specific programming styles of DBMS may be exploited to design more adequate interfaces.

Definition of scalability measures: the definition of a synchronous, predictable TCB environment will impose severe scalability restrictions if not conveniently addressed. Several measures, like multiplexing or merging, might have to be defined and applied.

5. Planned experiments

To extend DBMS in order to support transactions with TACID properties, the transactional engine of the DBMS has to be modified to include the detection of timing failures.

For the evaluation of the feasibility of timeliness transactions on database applications we are planning to use the PostgreSQL DBMS. The PostgreSQL DBMS is one of the most complete open-source databases available (it is one of the few open-source databases that support transactions). PostgreSQL is a small-size DBMS that is frequently used to support non-critical applications. For these reasons, we have chosen this DBMS as case study to figure out ways to add the timeliness property to the transactions supported by this DBMS.

The TCB will be used (and maybe included as part, or as an oracle of the DBMS) to detect timing failures and notify the client applications about the occurrence of this type of failures.

References

[1] R. Ramakrishnan, "Database Management Systems" second edition, McGraw Hill, ISBN 0-07-232206-3, 1999.

[2] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks ", Communications of the ACM, 1970.

[3] J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques", The Morgan Kaufmann Series in Data Management Systems, Jim Gray, 1993.

[4] K. Ramamritham, "Real-Time Databases", International Journal of Distributed andParallel Databases, 1996.

[5] P. Veríssimo and A. Casimiro. "The Timely Computing Base model and architecture". Transactions on Computers - Special Issue on Asynchronous Real-Time Systems, 2002.

[6] A. Casimiro, P. Martins and P. Veríssimo. "How to Build a Timely Computing Base using Real-Time Linux". In Proc. of the 2000 IEEE Workshop on Factory Communication Systems, pp.127–134, Porto, Portugal, September 2000.

Acknowledgements

Funding for this paper was provided, in part, by Portuguese Government/European Union through R&D Unit 326/94 (CISUC) and by the European Union, through DBench project, IST 2000 - 25425 (DBENCH) and Cortex project, IST-FET-2000-26031 (CORTEX).