TACID Transactions

Marco Vieira
CISUC - University of Coimbra
Coimbra, Portugal
mvieira@dei.uc.pt

António Casimiro Costa
FC - University of Lisbon
Lisbon, Portugal
casim@di.fc.ul.pt

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

Abstract

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, 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.

1. Introduction

On time data management is becoming a key difficulty faced by the information infrastructure of most organizations. A major problem is the capability of database applications to access and update data in a timely manner.

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 (DBMS). Most DBMS today are based on the relational data model proposed by E. F. Codd in 1970 [2]. The relational data model is very simple and defines a database as a collection of 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) 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. Figure 1 presents a simplified view of a typical database environment.

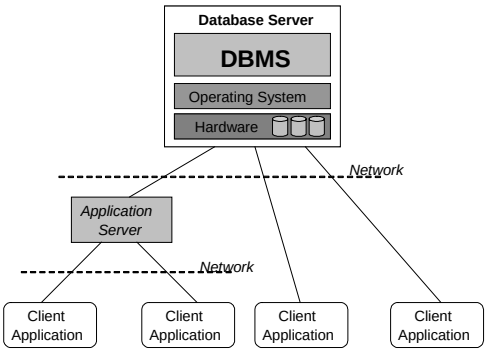

Figure 1. Typical database environment.

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 (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.
- **Durability:** the effects of a transaction that has been successfully completed must persist, even when the system has a failure after transaction finishing.

Database applications for critical areas (e.g., air traffic control, stock market, factories control, etc.) are increasingly giving more importance to the timely execution of transactions. However, developing database applications with timeliness requirements is a very 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, typical DBMS, such as Oracle, IBM DB2, MS SQLServer, PostgreSQL, etc, do not assure any temporal property, not even the detection of the cases when the transaction takes longer than the expected/desired time.

The DBMS is the backbone component in any transactional environment. Although a transactional environment can be composed by several layers (e.g., client application, web-server, application-server, DBMS, etc.) the DBMS is always the ultimate layer responsible for guaranteeing the execution of transactions according to the ACID properties. In a similar way, the DBMS is also the key component in which on time data management has to be implemented/guaranteed.

Our idea is to add timeliness properties to the typical ACID properties supported by transactions in most DBMS, in order to provide to the application layer timely ACID properties or, in short, TACID.

The structure of this paper is as follows. The following section presents our current view on TACID transactions. In section 3 we present some important aspects on timely computing. Section 4 presents the planned research approach and section 5 concludes the paper.

2. Timely ACID Transactions

As mentioned before, DBMS have a long tradition in high dependability, particularly in what concerns to data integrity and availability aspects. However, and in spite of the importance of time properties in some applications domains, there are no mechanisms implemented in typical commercial DBMS able to provide timeliness guarantees in the execution of transactions or at least to detect any type of timing failures.

In many situations timeliness is as important as correctness, which means that (approximate) correctness can be traded for timeliness [4]. Similarly, atomicity aspects may be relaxed. For instance, in a database application designed to manage information about a critical activity (e.g., air traffic control), a transaction that reads and stores the current reading from a positioning radar must be executed in a short time (i.e., the longer it takes to execute the transaction the less useful the reading becomes). In other words, in many database applications, when a transaction is submitted and it does not complete before a specified deadline that transaction becomes irrelevant and this situation must be reported to the application/business layer in order to be handled in an adequate way.

Real-time databases have emerged some years ago. However, these databases have been designed and implemented for very specific applications [4, 5]. To better support real-time applications, real-time databases redefine the ACID requirements to allow better support for temporal consistency while maintaining support for logical consistency [6]. These definitions use semantic information to determine to what degree the ACID properties must be enforced.

In real-time DBMS the ACID properties are applied only to parts of the transaction. Nevertheless, important features such as timing failure detection or, more generically, timing fault-tolerance, have been completely neglected, which also restricts the application areas for such DBMS. The problem is even worse if we consider the possibility of deploying distributed DBMS over wide-area or open environments. Such environments exhibit poor baseline synchrony and reliability properties, thus making it more difficult to deal with timeliness requirements. Obviously, this uncertainty and lack of timeliness will directly affect the execution of transactions, which, as an immediate effect, will be delayed. However, more severe effects may also be observed on the account of timing failures.

Our proposal is to bring timeliness properties to the typical ACID transactions implemented by most commercial DBMS, putting together classic database transactions management and recent achievements in the field of real-time and distributed timely computing. The goal is then to extend typical DBMS in order to support transactions with TACID (Timeliness, Atomicity, Consistency, Isolation, and Durability) properties. Thus, three classes of transactions will be considered:

1. **Transactions with no temporal requirements** (typical ACID transactions).

2. **Transactions with strict temporal requirements** (TACID transactions): for this class the database clients will be able to specify a time frame in which the transaction has to be concluded to succeed. In this class, the system must at least provide timing failure detection, including in distributed transactional environments.

3. **Transactions with probabilistic temporal requirements** (relaxed TACID transactions): in this class the transactions are always executed independently of time frame specified. However, a probability for the execution of a given transaction within that time frame will be provided to the client application before starting the transaction.

It is worth noting that the transaction class mentioned in 3 (relaxed TACID transactions) can also include timing failure detection capabilities such as the ones provided by transactions with strict temporal requirements. Another important aspect is that the profiles of all the transactions used in a given application must be known by the system (which is always the case in real database applications) in order to allow the probabilistic estimation of the transaction execution time before actually executing the transactions.

The estimation of the execution time (and cost in general) of SQL commands based on statistics previously collected is a quite mature topic in the database area [1]. However, this is used for read commands (basically the Select command) and not for the data manipulation commands that matter in transactions, as we propose as research approach in this paper.

The TACID transactions mechanisms should be provided to database programmers in the form of a programming interface or, better, by extending the typical transactional SQL commands.

3. Timely Computing

To address the problem of uncertainty, one possibility is to employ a design philosophy based on the assumption that the system is not homogeneous and has some parts which are more predictable than others. Combining this assumption and a proactive approach to further enforce the achievement of such predictability when and where necessary, we obtain a sort of shortcuts, or *wormholes*, through which it is possible to do things much faster or reliably than apparently possible in the other parts of the system.

This wormhole concept is generic and can in fact be instantiated in different ways. For example, when applied in the security domain, a wormhole takes the form of a security kernel, a trusted component as described in [7]. On the other hand, when timeliness is the relevant non-functional property to secure, the wormhole should essentially be timely.

An appropriate instance of such timeliness wormhole, which may be used as the framework for the problem presented in this paper, is the Timely Computing Base (TCB) model [8]. Essentially, a system with a TCB is divided into two well-defined parts:

- 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 wormhole, 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 has to provide a set of minimal services and must define a payload-to-TCB interface. Previous TCB implementations [9] provided the following services: **Duration Measurement**, **Timing Failure Detection** (TFD) and **Timely Execution**.

When considering the specific case of DBMS with TACID properties, it is important to understand which are the new challenges that must be dealt in order to use a TCB. As a result of our initial investigations, we identified the following challenges:

- **Definition of TCB services:** in order to optimize and simplify the TCB, 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 (e.g., SQL commands) 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.

4. Research Approach

To implement transactions with TACID properties, the transactional engine of DBMS has to be modified. A possible approach is to use the TCB model (and maybe include it 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.

The following subsections present an overview on how we are planning to address the problem of implementing TACID transactions in DBMS.

4.1. Identification of the requirements of timed transactions

The Timely Computing Base (TCB) model [8] provides a generic solution, a timeliness wormhole, for timely computing in environments of uncertain synchrony. It defines both the architectural constructs needed to address the problem and a programming model suited for a large range of applications.

We believe that the TCB approach can be applied in the context of this research in order to construct timed transactional systems in partially synchronous environments.

However, since TACID transactions are related to a specific application type, it is fundamental to investigate whether the programming model and the generic interfaces for interacting with a TCB wormhole might be specialized, in order to take full benefit of the TCB approach. For example, the services provided by a TCB are designed to handle arbitrary temporal specification, while in a (timed) transactional system it might be possible to restrict the possible range of specifications or even enforce all transactions to respect fixed temporal bounds. The possibility of applying this kind of restrictions will have a clear impact on the effectiveness of the solution and, possibly, on the scalability of the system (extremely important for DBMS).

In order to identify all the specific characteristics that might be used to design a refined and DBMS specific TCB wormhole (to which we will call DBMS wormhole), we will need to investigate the characteristics of transactional systems. Some important aspects that need to be addressed are:

1. Selection of typical transactional applications with timeliness requirements, which will be used as case studies to derive the relevant characteristics for the purposes of the study.
2. Identification of specific characteristics and requirements (with respect to the temporal behavior) of the selected example systems, which will be organized in order to find those that are common to all of them.
3. Translation of high-level timeliness requirements into low-level ones. A TCB timeliness wormhole provides just the basic services related to observing and securing timeliness properties. Therefore, it is necessary to understand how high-level requirements (for example, a temporal bound for the completion of a complex transaction) translate into simple requirements (timeliness bounds observable by the TCB wormhole).

4.2. Analysis of DBMS core implementations

Open source DBMS (e.g., PostgreSQL, MySQL, etc.) are increasingly being used to support the operations of organizations. These DBMS are obviously the candidates for case study to demonstrate the TACID concepts.

The goal is to analyze the implementation of several open source DBMS in order to better understand how transactional engines work. The study of the implementation of DBMS is a very important aspect to understand how DBMS are implemented, identify possible hooks for plugging timing constraints, and select an adequate DBMS for the prototype that we are planning to implement.

4.3. Definition of a DBMS wormhole

The concept of wormhole-based systems has already been applied for the definition of a generic timeliness wormhole (the Timely Computing Base, TCB) and a generic security/timeliness wormhole (the Trusted Timely Computing Base, TTCB). However, although there exist a publicly available prototype implementation of a TCB wormhole (http://www.navigators.di.fc.ul.pt/software/tcb/), this was designed as a proof-of-concept prototype, for experimental purposes, without taking into account potentially stringent or specific requirements of particular application classes.

In TACID we consider the specific case of DBMS with timeliness requirements. Therefore, we need to design a wormhole well suited for DBMS, a *DBMS wormhole*, which will be used to achieve the desired TACID properties.

Our aim is to explore, enhance and adapt the generic approaches to the construction of dependable and timely systems using a TCB wormhole, for the case of DBMS using a DBMS wormhole. Example application classes that may be constructed with a TCB include:

- **Fail-safe class**: applications that can switch to a fail-safe state when there is a timing failure.
- **Time-elastic class**: applications that are able to adapt timing constraints during execution.

In our research we are planning to focus on both approaches (the fail-safe and the time-elastic) to handle TACID transactions with the help of the defined DBMS wormhole.

In the former case (fail-safe class), we will consider TACID transactions whose timing bounds have to be always secured, or else the transaction has to be rolled back and the client application must be notified. This approach is expected to impose severe requirements to the wormhole and to its ability to detect every possible failure, which will require possibly some performance evaluation.

In the latter case (time-elastic class), we will consider a transactional system where transactions may commit after a given deadline thus producing timing failures, and the objective will be to adapt the system (for instance, by adjusting deadlines or by limiting the number of concurrent transactions) in order to ensure that the probability of delayed TACID transactions will stay below a specified value.

4.4. Implementation and evaluation

As a concretization of our research we are planning to define and develop a prototype in order to validate the concepts and assure that the outcomes of this research represent a practical and meaningful characterization of the timeliness requirements in database applications, both from the users and the system developer's point of view.

The evaluation of results will allow to push further the enabling technologies and to effectively define the concepts and prototype.

Performance and dependability benchmarks [10, 11, 12] are probably the best tools for the prototype validation. The study of benchmarking measures for systems with temporal requirements and the analysis of performance overhead introduced by timing specifications will also be addressed.

5. Conclusion

This paper presents a new problem related to timely execution of database transactions. Although timeliness requirements are of utmost importance in some database application areas, typical commercial DBMS do not assure any temporal property.

The idea presented in this paper consists in adding timeliness properties to the typical ACID properties supported by most DBMS. This paper presented the problem and the intended re-search approach to solve that problem.

6. References

[1] R. Ramakrishnan, “Database Management Systems” 2nd Ed., 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 and Parallel Databases, 1996.

[5] G. Ijzsoyoilu, R. T. Snodgrass, “Temporal and Real-Time Databases: A Survey”, IEEE Transactions On Knowledge and Data Engineering, 1995.

[6] L. DiPippo, V. Wolfe, “Real-Time Databases”, Database Systems Handbook, Multiscience Press, 1997.

[7] M. Correia, P. Veríssimo, and N. F. Neves. “The design of a COTS real-time distributed security kernel”, Fourth European Dependable Computing Conference, October 2002.

[8] P. Veríssimo, A. Casimiro, “The Timely Computing Base model and architecture”, Transactions on Computers-Special Issue on Asynchronous R-T Systems, 2002.

[9] A. Casimiro, P. Martins and P. Veríssimo, “How to Build a Timely Computing Base using Real-Time Linux”, 2000 IEEE Workshop on Factory Communication Systems, pp.127–134, Porto, Portugal, Sep. 2000.

[10] Transaction Processing Performance Consortium, “TPC Benchmark C, Standard Specification”, 2002.

[11] M. Vieira and H. Madeira, “A Dependability Benchmark for OLTP Application Environments”, 29th International Conference on Very Large Data Bases, VLDB2003, Berlin, Germany, Sept., 2003.

[12] M. Vieira and H. Madeira, “Benchmarking the Dependability of Different OLTP Systems”, IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-DCC2003, San Francisco, CA, June 22 - 25, 2003.