Trading-off Data Consistency for Timeliness in Real-Time Database Systems

Simin Cai, Barbara Gallina, Dag Nyström, Cristina Seceleanu
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
{simin.cai, barbara.gallina, dag.nystrom, cristina.seceleanu}@mdh.se

Abstract—In order to guarantee transaction timeliness, Real-time Database Management Systems (RTDBMSs) often relax data consistency by relaxing the ACID transaction properties. Such relaxation varies depending on the application and thus different transaction management mechanisms have to be decided for developing a tailored RTDBMS. However, current RTDBMSs development does not include systematic verification of timeliness and desired ACID properties. Consequently, the implemented transaction management mechanisms may breach timeliness of transactions. In this paper, we propose a process called DAGGERS for developing a tailored RTDBMS that guarantees timeliness and desired data consistency for real-time systems by employing model-checking techniques during the process. Based on the characteristics of the desired data manipulations, transaction models are designed and then formally verified iteratively together with selected run-time mechanisms, in order to achieve the desired/necessary trade-offs between timeliness and data consistency. The outcome of DAGGERS is thus a tailored transaction management mechanism with guaranteed appropriate trade-offs, as well as the model-checking based worst-case execution times and blocking times of transactions under these mechanisms and assumptions of the hardware architecture.

Keywords—RTDBMS, timeliness, ACID, formal verification.

I. INTRODUCTION

Real-time Database Management Systems (RTDBMSs) often serve as data management middleware for Real-Time Systems (RTSs), providing consistent data manipulation and guaranteed transaction timeliness. Traditionally, data consistency is the most important concern in non-real-time Database Management Systems (DBMSs), and ensured by transaction management. A transaction must guarantee the so-called ACID properties, that is, atomicity (a transaction either runs completely orrollbacks all changes), consistency (a transaction executing by itself must not violate logical constraints), isolation (uncommitted changes of one transaction should not be seen by concurrent transactions) and durability (committed changes are made permanent) [1]. In an RTDBMS, however, timely response of transactions must be guaranteed, even at the cost of data consistency [2]. In order to ensure timeliness, RTDBMSs often need to relax the ACID assurance [2].

The relaxation of ACID may differ from application to application, since many RTSs are implemented as special-purpose embedded solutions and thus entail high variability in both functional and extra-functional data management requirements [3]. Consequently, the designer of an RTDBMS must decide the specific run-time mechanisms for a particular real-time application, for instance the appropriate concurrency control mechanism and recovery mechanism, which ensure the desired/necessary relaxation of ACID.

However, during the development of RTDBMS, there exists no systematic method that guides the decisions of these run-time mechanisms so that the timeliness is guaranteed and the desired ACID relaxation is achieved. This may lead to an unpredictable system as well as unnecessary trade-offs. On the one hand, the decision of such mechanisms will inevitably impact the response time of transactions and may even result in unbounded execution or blocking time. Let us assume that, in order to ensure desired isolation, one needs to implement a concurrency control mechanism WAIT-50 [4] in the RTDBMS. This might introduce unbounded delay caused by transaction rollbacks and restarts due to conflicts. On the other hand, it is difficult to reason and verify if the relaxation of ACID is truly necessary. Consequently, we need an RTDBMS design process that could solve such challenges.

In this paper, we propose the DAGGERS process for developing a tailored RTDBMS that ensures transaction timeliness and desired data consistency. Our process allows system designers to systematically specify the trade-offs between global logical consistency and temporal constraints of transactions, from system requirements, and create a specialized transaction model with the desired properties. In order to ensure such properties, we carry out formal verification of transactional behaviors against the desired properties, which are formalized together with candidate run-time mechanisms for transaction management within the UPPAAL environment [5]. Finally, the selected run-time mechanisms that are proven to solve possibly existing conflicts are woven into the RTDBMS.

Our main contribution is two-fold. First, we provide the run-time mechanisms to tailor the RTDBMS such that it ensures the desired relaxation of ACID without breaching timeliness. Second, we provide the real-time properties such as worst-case execution time (WCET) and blocking time of the transactions, which account for the selected mechanisms in the final outcome under particularly stated hardware assumptions.

The paper is structured as follows: Section II describes the system model. The process is presented in Section III. The technical challenges, as well as the status of our research, are presented in Section IV. In Section V we present the related work. Finally we summarize the paper.

II. ASSUMED SYSTEM MODEL

In this paper we consider a hard real-time system where all data access and manipulation go through transactions in an RTDBMS.

Logically related operations on data form work units. Each work unit WUi includes a set of read and write operations in the database, as well as operations that perform calculation on data. Each work unit has a set of real-time requirements, including its invocation pattern and expected response time.

Each work unit WUi is encapsulated as a transaction TRi in the RTDBMS, associated with the desired ACID properties to be ensured. Each transaction also has a set of real-time properties, with a period (or minimal inter-arrival time) and
a relative deadline inherited from the work unit. The ACID assurance of transactions requires transaction management mechanisms to start, abort and rollback the transactions in a controlled manner, which in return affects the execution time and blocking time. The execution time of a transaction includes the execution time of operations in the work unit, plus the execution time of computation to ensure the ACID properties, such as lock acquisition and logging. A transaction may be blocked by concurrent transactions. The blocking time is determined by the concurrency control mechanism implemented in the RTDBMS.

At the scheduler level, transaction TR$_i$ is executed as a task $\tau_i$. A scheduling policy is adopted by the scheduler, which is not part of the RTDBMS. In order to achieve a predictable system, the execution time and blocking time have to be bounded, and the task set should be schedulable. These constraints, together with the desired ACID properties, form the constraints on the decision of run-time mechanisms implemented in the RTDBMS.

### III. THE DAGGERS PROCESS

Our DAGGERS process is a design-time systematic process based on formal methods, meant to support system engineers in identifying, designing, and implementing adequate transactional support for RTDBMS in real-time systems. DAGGERS inherits the PRISMA [8] terminological framework as well as its methodological approach. In this section, we present the major steps of the DAGGERS process, as shown in Fig. 1. In the figure, A, C, I, D and T stand for atomicity, consistency, isolation, durability and timeliness, respectively. A specific relaxation of atomicity, also called a variant of atomicity [6], is denoted as $A_i$, while one specific instance of $A_i$ is denoted as $A_{i,k}$. Similar denotations are applied to C, I, D and T variants. In this paper we focus on hard real-time systems, in which timeliness should never be relaxed (thus no other variants of T).

In principle the process can be applied to mixed criticality RTSSs, where multiple variants of timeliness may exist.

We start with specifying the operations in the work units and their logical and temporal properties. Then we formalize them into transaction models, and verify the timeliness and desired ACID properties with selected run-time mechanisms. Finally, under particular hardware behavior assumptions/models, we obtain a set of tasks with their WCETs and blocking times, which can be used for schedulability analysis, as well as the verified run-time mechanisms for a tailored RTDBMS.

In the following we describe the steps of the process.

**Step I:** From system requirements a data management expert identifies the work units, which are partially ordered sets of logically related operations on data items, as well as the logical and temporal requirements characterizing these work units. The initial transactional properties are then specified based on these requirements. A set of desired properties is associated to each work unit, respectively, consisting of certain variants of ACID and the timeliness constraints. For better illustration, we assume that the desired isolation variant I$_k$ is identified to be PL-3 isolation level, which ensures full isolation [7]. These specified computational behaviors together with the properties are called transaction types.

**Step II:** The second step of DAGGERS consists of the building, iterative tailoring and refinement of the transaction models. First, by analyzing the dependencies between transaction types, the data management expert organizes the transaction types according to well-known transaction model types. Next, initial yet formal design specifications are created, including the design-level decisions for the ACID variants, such as an optimistic concurrency control algorithm for PL-3 isolation, denoted as I$_k$. The specifications can then be modeled and verified by model-checking tools.

We propose to model transactional behavior as networks of timed automata [8], which have underlying formal semantics implemented in the state-of-the-art model-checker for real-time systems, called UPPAAL [5]. In order to find possible conflicting behaviors, we model-check the transaction models annotated with timing information in form of invariants. The output of the verification is given in terms of “yes/no” answers, but also in terms of timing properties of transactions (e.g., execution time, blocking time, etc.). In case of a “no” answer (meaning that inconsistencies exist) the verification returns a counter-example that exposes the respective inconsistency. The verification and validation of the transactions can be used to detect: (i) any behavioral inconsistencies between the stated transactional properties and the logical data requirements, as well as (ii) any unresolvable timing inconsistencies between the potential conflicts among concurrently executed transactions and the timing requirements.

The transaction models are checked iteratively, together with different candidate run-time mechanisms, provided by the platform. If model-checking shows that unsolvable conflicts occur with a particular candidate run-time mechanism (e.g., optimistic concurrency control), the mechanism will be replaced by another candidate (e.g., a lock-based algorithm), and the model is verified again. Ideally, the successive checking stops when no conflicts are detected anymore.

In case inconsistencies that cannot be resolved by any candidate run-time mechanism model exist, traces of the inconsistency as well as other feedback from the verification will be stored into a repository. The obtained feedback can act as heuristics that may help the data management expert to change the transaction models, adjust the scheduling policy to separate such transactions, or reflect further on the requirements. For instance, a less restrictive isolation variant such as PL-1 isolation level [7] might be considered to replace the PL-3 isolation, since deadlines will be missed if the system has to ensure the current logical consistency requirements. Then the modeling and model-checking will restart. This iteration continues until a consistent formalization is achieved.

At the end of this step, a set of refined transaction models is generated, with specified timing constraints such as transaction deadlines, and appropriate variants of atomicity, consistency, isolation and durability. The WCET and blocking time for each transaction are obtained with these selected mechanisms, under particular hardware assumptions (e.g., we can assume a hardware whose impact on the WCET is so small that it can be ignored, or we can use abstract hardware architecture (including cache) models in UPPAAL). In addition to transaction models, the output of this step also includes the potential conflicts, and the corresponding run-time mechanisms able to resolve these conflicts, for each transaction model respectively.

**Step III:** In this step, we form a task set with their WCETs and blocking times. In addition, we automatically generate an RTDBMS by composing the selected run-time mechanisms. We propose to generate the RTDBMS using the COMET (COMponent-based Embedded Real-Time) [9] platform, which consists of a set of components that encapsulate specific real-time database functionalities, and a set of aspects that encapsulate cross-cutting concerns such as concurrency control.
and logging. By weaving selected components and aspects from COMET, we generate a tailored RTDBMS that provides run-time support for the resolution of potential conflicts and assurance of desired transactional properties.

IV. TECHNICAL CHALLENGES AND STATUS OF RESEARCH

Several key challenges need to be addressed in order to support the DAGGERS process. One major challenge is specifying and modeling the real-time transaction models. The ACID and timeliness properties should be specified, in such a way that we can easily reason about the possible relaxation of ACID for timeliness. Existing techniques either only focus on ACID, or have limitation in formal syntax and semantics. The properties need to be specified in Timed Computation Tree Logic (TCTL), and the behaviors of transactions as networks of timed automata in UPPAAL. During the iterative verification step, different candidate run-time mechanism models need to be woven into the timed automata network dynamically, which is not a trivial task. Guidelines and tool support, for example specification patterns [10], are desired for the correctness and efficiency of modeling.

Obtaining the WCETs and blocking times of transactions is another challenge. Although some work has been done, for example by Gustavsson et. al [11], on WCET analysis for concurrent programs using UPPAAL, it remains nontrivial to model and analyze database transactions with ACID properties.

Another challenge is to find a predictable resolution mechanism for the detected conflicts. In some situations, conflicts may be resolvable without missing any deadlines if transactions with certain dependencies are grouped together and different run-time mechanisms are applied on different groups. Methods for grouping transactions are necessary, which are performed by either the verification tool, or the database expert. It is also possible that the conflicts among some transactions are not resolvable within bounded time by any run-time mechanisms. For instance, no concurrency control mechanism can ensure the desired isolation of particular transactions without breaching their deadlines. In this case, these transactions have to be separated in time, using techniques such as static scheduling [12], offset scheduling [13] or explicit mutual exclusion by semaphores in the transactions. In order to adjust the scheduling policy, the verification framework should provide traces of the conflicts, as well as other information such as the blocking time from conflicting transactions and rolling-back/restarting time caused by the conflicts.

One common challenge for solutions using model-checking techniques is the scalability of model checking. UPPAAL-SMC, an extension of UPPAAL with statistical model checking for priced timed automata [14], improves the scalability via bounded model-checking and can be used for the bounded verification of complex systems, assuming certain probability distributions. However, the provided result is not a guarantee as in the symbolic model-checking, but rather a probability estimation of a certain property, with a specific accuracy.

Regarding the status of our research, we have already carried out some preliminary work on modeling transactions with particular concurrency control mechanisms in UPPAAL [15], and will continue to involve other mechanisms and properties. Currently we are also working on a taxonomy of real-time data aggregation, with a focus on the trade-offs between transaction properties during the aggregation process. In our next steps, we plan to develop a formal language for specifying real-time transaction properties from system requirements. We will then develop a framework based on UPPAAL, including models of common candidate run-time mechanisms, for building and iterative verification of transaction models. Finally, we plan to implement an RTDBMS with the tailored transaction management using the COMET platform.

V. RELATED WORK

Relaxing ACID for a compromised consistency has been investigated in the database community. In recent work, nec-
ecessary trade-offs have been proposed among data consistency, availability and scalability [16], as well as database performance [17]. However, existing works do not take timeliness into consideration, which is a core property for RTSs and the focus of our process.

Noticeable research efforts have been made in engineering tailored DBMSs in recent years. COMET [9] combines a component-based approach and aspect-oriented programing to build tailored RTDBMSs. Encapsulating database functionalities as components, and crosscutting features as aspects, COMET generates a tailored RTDBMS by weaving the selected components and aspects together. In FAME-DBMS [11], functional requirements on a DBMS are represented as features. DBMS variants are generated by composing the reusable features. The selection of building modules in these approaches is based on functional requirements analysis, as well as constraints on code-size and performance. They mainly address resource consumption and footprint issues for embedded systems, rather than the timeliness of transactions and the possible conflicts with ACID assurance. Our process, as a contrast, starts from deriving the real-time transaction models accounting for both timeliness and ACID properties. During the verification of transaction models, the run-time implementations are selected, and are proved to guarantee the desired timeliness and data consistency requirements. The existing platforms for generating the tailored DBMS can be integrated into our process at the implementation phase.

Substantial work has been done on formal specification and reasoning of transaction models. The ACTA [18] framework provides a first order logic formalization to specify the transactional effects on data and the interaction between transactions, facilitating reasoning of transaction properties and flexible synthesis of transaction models. Real-Time ACTA [19] extends ACTA with formalization of real-time constraints on transactions and data. However, the formal syntax and semantics for specification of ACID variants provided by ACTA and Real-Time ACTA are limited, and tool-support for verification is lacking. SPLACID [20] improves ACTA by providing a more complete language support for ACID variants and their sub-features, but real-time properties are not included. Non-ACTA descendant work has also been proposed. For example, Wang et al. [21] proposed Abstract Transaction Construct (ATC) that encapsulates the structure and behavior of a transaction service. However, none of these works provide specification for both ACID and real-time properties, or support verification of transaction models with run-time mechanisms.

VI. SUMMARY

In this paper, we have introduced the DAGGERS process (of our newly funded project DAGGERS), which aims at generating a tailored RTDBMS that ensures appropriate trade-offs between global data consistency and transaction timeliness in RTSSs. In this process, work units and their ACID and timeliness properties are specified from system requirements. Based on this specification, transaction models are derived and iteratively refined by means of formal verification. By composing the run-time mechanisms proved to resolve possible detected conflicts, an RTDBMS is generated to support the refined transaction models. A task set with their WCETs and blocking times are obtained.

Compared to current tailored DBMS solutions that commonly base their customization on the functional requirements, and constraints on resource consumption, footprint and performance, our process relies on the analysis and verification of transaction models, thus achieving the desired data consistency and transaction timeliness required by the particular RTS.

REFERENCES


