A self-adaptable scheduler for synchronizing transactions in dynamically configurable environments

Maristela Holanda\textsuperscript{a,*}, Angelo Brayner\textsuperscript{b,1}, Sergio Fialho\textsuperscript{c,2}

\textsuperscript{a} Catholic University of Brasília, Department of Computer Science, 72030-170 Brasília, Brazil
\textsuperscript{b} University of Fortaleza, Department of Computer Science, 60811-341 Fortaleza, Brazil
\textsuperscript{c} Federal University of Rio Grande do Norte, Department of Computer Engineering, 59072-970 Natal, RN, Brazil

\textbf{A R T I C L E   I N F O}

Article history:
Received 27 October 2007
Received in revised form 28 February 2008
Accepted 28 February 2008
Available online 18 March 2008

Keywords:
Database
Concurrency control
Scheduler
Transaction and MDB

\textbf{A B S T R A C T}

In a database system, the scheduler has the goal of synchronizing operations belonging to several concurrent transactions. The scheduler implements a concurrency control protocol, which may have either conservative or aggressive behavior. Existing database systems have schedulers with static behavior. This paper presents a self-adaptable scheduler, called Intelligent Transaction Scheduler (ITS), which has the ability of dynamically changing its behavior to adapt itself to the characteristics of the computing environment, without any human interference by using an expert system based on fuzzy logic. ITS can implement different correctness criteria, such as classical (syntactic) serializability and semantic serializability.

1. Introduction

Users interact with database systems by means of application programs. A transaction is an abstraction that represents a sequence of database operations resulting from the execution of an application program [28]. A database system should synchronize the execution of concurrent transactions to guarantee database consistency. The component in the database system responsible for synchronizing the operations of concurrent transactions is the scheduler. The scheduler synchronizes operations belonging to different transactions by means of a concurrency control protocol.

Concurrency control protocols may present aggressive or conservative behavior [28]. An aggressive concurrency control protocol tends to avoid delaying operations, and tries to schedule them immediately. However, such a strategy may lead to a situation in which there is no possibility of yielding a correct execution of all active transactions. In this case, operations of one or more transactions should be rejected. A conservative protocol, on the other hand, tends to delay operations in order to synchronize them correctly.

Additionally, aggressive concurrency protocols can be categorized in two sub-classes: pessimistic and optimistic. A scheduler implementing a pessimistic protocol decides to accept or reject the operation as soon as it receives the operation. On the other hand, a scheduler using an optimistic protocol immediately schedules the operation. After a given period of time, the scheduler verifies the correctness of the schedule it has already produced. If the schedule is correct, the scheduler continues synchronizing operations. Otherwise, the scheduler must abort some transactions.
The concept of serializability [18] has been used since the 1970s by concurrency control protocols as a correctness criterion to ensure database consistency. The Two Phase Locking Protocol (2PL) is the most widely used conservative protocol based on serializability. Among the protocols with aggressive behavior that use a pessimistic approach, the Timestamp Ordering (TO), and the Serialization Graph Testing (SGT), are noteworthy. A scheduler implementing an aggressive concurrency control protocol is called an aggressive scheduler, whereas a scheduler implementing a conservative concurrency control protocol is called a conservative scheduler.

A Mobile Ad hoc NETwork (MANET) [16] is a wireless dynamic network, in which nodes can join or leave the network dynamically. Some database applications that use MANET technology are: industrial and commercial applications involving cooperative mobile data exchange [16]; military networks [21,26]; vehicular ad hoc networks [9]; city services, such as the taxi company system presented in [12]. MANETs allow users carrying portable devices to access database services regardless of their physical location or movement patterns. Thus, a dynamic collection of autonomous mobile databases, called mobile database community (MDBC) [3], can be formed, where the mobile databases reside in nodes (Mobile Units – MU) of a MANET. Since new participants may join an MDBC or a participant may transiently disconnect from the MDBC (due to communication disruptions or to save power), an MDBC is characterized as a dynamically configurable environment. Observe that the concept of MDBC models a dynamically configurable multidatabase [6] whose members (local databases) can move across different locations.

As already mentioned, mobile units may join and leave an MDBC at anytime. In such a dynamic environment, the use of a fixed concurrency protocol (e.g., using only strict 2PL) to generate correct schedules may not be the most efficient strategy at given moments over a period of time. This is because a scheduler with static behavior is not able to capture modifications in the context it is being executed. As shown in Section 4, the arrival of a new mobile unit in an MDBC drastically modifies the scenario in which transactions are being executed. For example, consider that the new mobile unit submits transactions with several write operations and a scheduler implementing a conservative protocol (e.g., strict 2PL) is being used. In such a scenario, the conflicting operation rate may increase (see Section 4), which may induce low throughput rates (low degree of parallelism).

This paper presents the Intelligent Transaction Scheduler (ITS), characterized by having a hybrid behavior (conservative or aggressive). The gist of this approach is to provide a scheduler that can automatically identify and react to the modifications in the computational environment in which it is inserted, by adapting its behavior (from aggressive to conservative and vice versa), without generating incorrect global schedules in dynamically configurable environments. Furthermore, the ITS can be adjusted to use correctness criteria other than classical serializability, since that criterion is very restrictive for controlling concurrency in dynamically configurable environments, such as an MDBC. Therefore, we claim that an adaptive scheduler is quite appropriate for controlling concurrency in dynamically configurable environments, whose configuration (topology) may change dynamically.

It is worthwhile to note that although the ITS has been developed for use in dynamically configurable environments, it can also be used to synchronize operations of concurrent transactions in any other distributed environment with static topology as well.

The remainder of this paper is organized as follows: Section 2 presents the architecture of the proposed scheduler (ITS); Section 3 presents the theoretical support which proves the correctness of ITS’ self-adaptability (i.e., ITS produces correct schedules); Section 4 describes aspects of ITS’ implementation, its use in MDBCs and an analysis of the obtained results; Section 5 describes some of the existing related proposals in the literature; and finally, Section 6 concludes this paper.

2. ITS – intelligent transaction scheduler

The ITS [24,25] is an intelligent scheduler and its architecture is composed of two modules, as shown in Fig. 1: the Analyzer, a component that makes decisions related to the most appropriate scheduler behavior according to characteristics of the computing environment during a given time period; and the Scheduler, the component that executes the concurrency control protocols selected by the Analyzer.

2.1. Analyzer

The Analyzer is an Expert System – ES based on fuzzy logic [19]. The Analyzer module presented in Fig. 1 illustrates the basic components of the ES: Knowledge Database, which stores the specialized knowledge used in decision making; Context, which keeps the values of the variables used to solve the problem; Inference Engine, which infers knowledge by means of the rules stored in the Knowledge Database and the variable values stored in the Context; and Interface, which controls the dialogue between the user (Scheduler) and the ES. The Interface receives the information sent by the Scheduler, stores it in the Context and sends the information about the decision made internally back to the Scheduler.

The ITS’ Analyzer was developed using the process presented in [7], which involves five steps. Each one of these steps is described as follows.

2.1.1. Specification of the problem and identification of input and output fuzzy variables

The problem is to choose the most appropriate scheduler behavior, either aggressive or conservative, according to a number of computing environment characteristics. The most appropriate behavior should reduce the delays caused by the synchronization of the operations by the scheduler, while keeping the aborted transaction rate within the lowest possible range.
During the knowledge acquisition phase, the following variables were investigated: read operation rate, write operation rate, conflicting operation rate, deadlock rate, aborted transaction rate, lock waiting rate, transaction size, initial scheduler behavior and database size. These variables were considered to influence the choice of the most appropriate scheduler behavior. After analyzing all of these variables, we realized that not all of them were necessary, since there are interrelationships between them. For example, read and write operation rates can be identified by using the conflicting operation rate; the deadlock rate can be inferred from the initial scheduler behavior and aborted transaction rate. For that reason, four input variables were chosen:

- **aborted transaction rate** ($atr$)
- **conflicting operation rate** ($cor$)
- **lock waiting rate** ($lwr$)
- **initial scheduler behavior** ($isb$)

The output variable is called **scheduler behavior** ($sb$), which defines the most appropriate behavior for the adaptive scheduler. The input variables used by ITS are described as follows.

- The $atr$ variable is the percentage of the number of aborted transactions in relation to all transactions. This variable is important because the ITS needs to guarantee the lowest aborted transaction rate for the specified scenario during a given period of time.
- The $cor$ variable is the percentage of the conflicting operations in relation to the total number of operations. This variable will help the Analyzer make a decision about the scheduler behavior, since the conflict among operations causes the delay of the execution and may cause an abort of the transactions.
- The $lwr$ variable is the percentage of the number of operations waiting for the lock to be released in relation to all operations of the conflicting transactions. This variable defines the delay generated by synchronization of the operations by the scheduler.
- The $isb$ variable, is used by the Analyzer to determine what behavior generated the aborted transaction rate, the lock waiting rate and the conflicting operation rate. The Analyzer is a separate module, and it does not know the internal scheduler behavior.

### 2.1.2. Definition of the fuzzy sets for each fuzzy variable

The fuzzy sets for the input variables $atr$, $cor$, $lwr$ were defined with a trapezoidal shape (Fig. 2). All these variables have the fuzzy values L (low), M (medium) and H (high), and their universe of discourse ranges from 0 to 100. The points $x_1$, $x_2$, $x_3$ and $x_4$ (Fig. 2) define the values L, M and H. These points should be provided by the application/database administrator (a specialist).

![Fig. 1. ITS abstract model.](image1)

![Fig. 2. Fuzzy Set for each linguistic variable: $atr$, $cor$, $lwr$, $isb$ and $sb$.](image2)
The trapezoidal shape was chosen for these fuzzy sets because this shape has good response time (linear function) and its configuration is easy. With this shape, the specialist defines the values he is certain of: interval \([0, x_1]\) where the variable presents a low value, the interval \([x_2, x_3]\) where it presents a medium value and the high value interval \([x_4, 100]\). The intervals in which the specialist is uncertain about behavior variations are represented by \([x_1, x_2]\) and \([x_3, x_4]\).

Fig. 2d illustrates the fuzzy sets for isb and sb variables, where the singleton function has just two values: A (aggressive) or C (conservative).

2.1.3. Construction of the rules

The knowledge acquisition process for the definition of the rules was developed by means of many meetings between the knowledge engineer and the database concurrency control specialists. The observation technique was used to analyze the synchronization process of the operations with fixed behavior schedulers. Several tests were executed with the fixed behavior schedulers to analyze the scheduler with different scenarios.

Many rules composed of the combination the input fuzzy variables were investigated, but some of them cannot occur in real environments. For example, the lock waiting rate for a scheduler with an aggressive behavior is 0, consequently the following combination is impossible: a scheduler with an aggressive behavior and lock waiting rate with a medium or high value. Behind these impossible combinations, some rules (combination of the input variable) can be reduced to just one rule.

This way, after analyzing all the combinations of the input variables, it was verified the 8 rules were sufficient.

After this process the knowledge database was structured with 8 rules described as follows:

**Rule 1:** If \( atr = L \) AND \( cor = L \) AND \( lwr = H \) THEN \( sb = A \). If the aborted transaction rate is low, the scheduler has already attained the best aborted transaction rate. If the conflicting operation rate is low, the lock waiting should also be low. However, if the lock waiting rate is high, most likely many operations are waiting unnecessarily, since the conflict rate is low. In this case, the scheduler should change its behavior to aggressive (output variable sb).

**Rule 2:** If \( atr = L \) AND \( cor = L \) AND \( lwr = M \) THEN \( sb = C \). If the aborted transaction rate is low (L), the scheduler has already attained the best aborted transaction rate. If the lock waiting rate is medium (M), then the scheduler behavior is necessarily conservative. Thus, the scheduler should not change its behavior because it is not possible to guarantee the same aborted transaction rate after the change of the scheduler behavior.

**Rule 3:** If \( atr = L \) AND \( lwr = L \) AND \( isb = C \) THEN \( sb = C \). If the aborted transaction rate is low (L), the scheduler has already attained the best aborted transaction rate. Moreover, if the lock waiting rate is low, then the ITS already has the appropriate behavior. Then, if the initial scheduler behavior is conservative, this behavior should continue.

**Rule 4:** If \( atr = L \) AND \( isb = A \) THEN \( sb = A \). The aggressive behavior ensures a high degree of concurrency. Therefore, if the scheduler already has that behavior and the aborted transaction rate is low, then it must continue with the aggressive behavior.

**Rule 5:** If \( atr = L \) AND \( cor = M \) AND \( isb = C \) THEN \( sb = C \).

**Rule 6:** If \( atr = L \) AND \( cor = H \) AND \( isb = C \) THEN \( sb = C \). For these two rules the following axiom is valid: if the aborted transaction rate is low (the scheduler already has the best value for this variable) and the conflicting operation rate is medium or high, independent of the lock waiting rate, the scheduler behavior must continue conservative. This is because it is not possible to guarantee that after a change to aggressive behavior the aborted transaction rate will continue low.

**Rule 7:** If \( atr = M \) THEN \( sb = C \).

**Rule 8:** If \( atr = H \) THEN \( sb = C \).

For rules 7 and 8 the following axiom is valid: if the aborted transaction rate is medium or high, the best scheduler behavior is conservative. If the scheduler is using the conservative behavior that delays the operations in order to avoid aborted transactions, then changing the scheduler behavior will most likely increase the aborted transaction rate. If the scheduler behavior is aggressive, producing medium or high value for the aborted transaction rate, then the scheduler should change its behavior to conservative to decrease the aborted transaction rate.

2.1.4. Development of the ES

The ES was implemented in Java language using the JFuzzy API [13] developed by National Research Council of Canada’s Institute for Information Technology. This API is a set of Java classes that provides the capability for handling fuzzy concepts and reasoning.

2.1.5. Evaluation and Improvement the ES

The ITS’s evaluation was executed during the entire specification, development and test phases of the ITS. During the specification some external specialists evaluated the linguistic variables and the rules. All rules are evaluated by means of the tests applied in the scheduler with fixed concurrency control protocols (aggressive or conservative).

Many tests were executed in relation to the linguistic variable values and their respective fuzzy sets, during the entire implementation phase, for guaranteeing that they are correct. In Section 4, the presented tests show that the Analyzer behaved as expected.
2.2. Scheduler

The module of the ITS responsible for synchronizing operations belonging to several transactions is the Scheduler. The scheduler is composed of the following components: Protocols, which stores the code of concurrency control protocols; Engine, which executes the concurrency control protocols; Aggressive DB and Conservative DB, which store necessary information for scheduling the transactions when the scheduler has aggressive or conservative behavior, respectively; Transition DB, which stores necessary information for controlling the transactions in the course of a transition period from one behavior to another; Interface, which implements the communication between the scheduler and the other components.

The ITS’ has three phases: Conservative Phase, Aggressive Phase and Transition Phase. During the conservative and aggressive phases all the transactions are scheduled using a concurrency control protocol with conservative or aggressive behavior, respectively. The transition phase represents the transition from one behavior to another. During the transition phase there might be transactions being synchronized by an aggressive protocol, while others are being synchronized by a conservative protocol. Accordingly, a key feature is to ensure that the ITS has the ability of producing correct schedules during the transition phase. In other words, the ITS should maintain database in a consistent state even when two concurrency control protocols are being executed simultaneously during the transition phase. In Section 3, we prove that the ITS presents that feature.

3. ITS' self-adaptability

The key feature of the ITS stems from the ability it owns to change its behavior dynamically in response to changes in the computational environment (e.g., a new client starts to submit several new transactions with write operations). In other words, ITS is able to switch the concurrency control protocol used for synchronizing operations belonging to concurrent transactions on-the-fly (i.e., without stopping its execution) and without human interference. For that reason, the ITS’ module implements concurrency control protocols with conservative and aggressive behavior.

Therefore, the challenge is to maintain the database in a consistent state during the transition phase, which corresponds to the time interval when there are transactions being synchronized by two different concurrency control protocols. This Section presents how ITS produces correct schedules, even during the transition phase and another important feature — it’s ability to implement two different correctness criteria: syntactic serializability (classical correctness criterion) and semantic serializability [4].

3.1. The transaction model

A transaction (Definition 1) represents a finite sequence of operations on database objects. The execution of a transaction is finalized with a commit or abort operation to indicate if the execution was successful or not, respectively [15,28]. The execution of concurrent transactions is modeled by the concept of schedule (Definition 2).

Definition 1 (Transaction). A transaction $T$ is a sequence of distinct actions $a_1, a_2, \ldots, a_n, a_i \in \{r, w, c, a\}$, where $r$ denotes a read operation and $w$ represents a write operation. The sequence is finalized with a commit (c) or an abort (a) operation. The set of operations of a transaction $T$ is represented by $OP(T)$.

Definition 2 (Schedule). A schedule $S$ over a set $T = \{T_1, T_2, \ldots, T_n\}$ of transactions represents an element of the shuffle product $T_1 \times T_2 \times \cdots \times T_n$. The order of operations belonging to a transaction $T_i \in T, 0 < i < n$, must be preserved by any schedule defined over $T$. That means, if an operation $p_i$ precedes $q_i$ in a transaction $T_i$ (i.e., $p_i < q_i$), then the execution of $p_i$ must happen before $q_i$ in any schedule over $T$. The set of operations of a schedule $S$ is represented by $OP(S)$.

Definition 3 (Projection). Let $S$ be a schedule over a set of transactions $T$ and $M$ a set of transactions, where $T \supseteq M$. The projection $P$ of $S$ on the set $M$ is a schedule for which the following conditions must hold:

- (i) $P$ only contains operations of transactions belonging to $M$;
- (ii) $\forall q \in OP(P)$, then $q \in OP(S)$, and
- (iii) $\forall o,q \in OP(P)$, $o < q$ then $\sigma_o o < q$.

Definition 4 (Conflicting operations). Let $p_i$ and $q_j$ be operations of transactions $T_i$ and $T_j$, respectively, with $i \neq j$. The operations $p_i$ and $q_j$ conflict, if at least one of them is a write operation, and both access the same database object.

3.2. ITS implementing classical serializability

The transition phase begins with the notification of a behavior change, which should be sent by the Analyzer to the Scheduler. This phase ends when transactions, which were active (see Definition 5) at the time of notification, become completed transactions (see Definition 6).
Definition 5 (Active transaction). A transaction \( T_i \) is active if:

\[ \forall o \in OP(T_i), \quad o \neq a \land o \neq c \]

Definition 6 (Completed transaction). A transaction \( T_i \) is completed if:

\[ \exists o \in OP(T_i), \quad o = a \lor o = c \]

The concept of serializability has been used to identify correct schedules. A schedule is said to be correct if it leaves the database in a consistent state. A schedule \( S \) over a set \( T \) of transactions is serializable if it is equivalent to some serial schedule over the set \( T \). Two schedules are equivalent if they produce the same final database state when executed over the same initial database state. There are several ways to define schedule equivalence, conflict equivalence being the most commonly used.

Two schedules are said to be conflict equivalent if the order of any two conflicting operations (see Definition 4) is the same in both schedules. In other words, two schedules \( S \) and \( S' \) over the set \( T \) of transactions are conflict equivalent if, for any conflicting operations \( p_i \) and \( q_j \) which belong to transactions \( T_i \in T \) and \( T_j \in T \), respectively, and \( p_i <_S q_j \) then \( p_i <_S q_j \). A schedule \( S \) is conflict serializable if it is conflict equivalent to some serial schedule. It is possible to check if the schedule is conflict serializable by means of its Serialization Graph. Let \( S \) be a schedule over the set of transactions \( T = \{ T_1, \ldots, T_n \} \). The serialization graph for \( S \) is a directed graph \( SG(S) = (N,E) \), where \( N = T \) and an edge \( T_i \rightarrow T_j \in E \) (with \( T_i, T_j \in T \) and \( i \neq j \)), if there are two operations \( p \in T_i \) and \( q \in T_j \) which are in conflict and \( p <_S q \). A schedule \( S \) is conflict serializable if \( SG(S) \) contains no cycle.

Before defining the notion of ITS schedule (a schedule produced by the ITS) and its correctness, it is necessary to define a new operation over schedules, called ordered join. The ordered join of two schedules, \( S \mid S' \), will result in a new schedule \( S' \), which contains an interleaved sequence of operations belonging to schedules \( S \) and \( S' \) and the order of the operations belonging to \( S \) and \( S' \) is preserved in \( S' \). The ordered join operation is defined formally as follows.

Definition 7 (Ordered join). An ordered join of two schedules \( S \) and \( S' \), \( S \mid S' \), results in a schedule \( S' \), where

1. \( OP(S') = OP(S) \cup OP(S') \)
2. \( \forall p,q \in OP(S) \land p <_S q \Rightarrow p <_S q \)
3. \( \forall p,q \in OP(S) \land p <_S q \Rightarrow p <_S q \land p <_S q \)
4. \( \forall p,q \in OP(S) \land p <_S q \Rightarrow p <_S q \land p <_S q \)

Fig. 3 illustrates an example of the ordered join operation between two schedules \( S \) and \( S' \). The schedule \( S \) is defined over the transactions \( T_1, T_2 \) and \( T_3 \), and \( S' \) is defined over the transactions \( T_4, T_5 \) and \( T_6 \). During a time interval \( i \), the schedule \( S' \) presents the interleaving of the operations of \( S \) and \( S' \), in accordance with the item 4 (iv) of Definition 7.

With this, it is possible to define the notion of an ITS, which corresponds to a complete schedule produced by ITS. Informally, an ITS is generated by applying an ordered join operation over three schedules, \( S_0, S_{transition} \) and \( S_i \), considering just one behavior change done by ITS. \( S_0 \) corresponds to the schedule defined over the set of transactions that begin their scheduling with the Protocol P1. \( S_i \) represents the schedule defined over the set of transactions submitted after the notification of behavior change (with new Protocol P2). Finally, \( S_{transition} \) is the schedule defined over the set of transactions that have operations synchronized during the transition phase.

Definition 8 (ITS). Let \( S_0 \) be a schedule on a set of transactions \( T^0 \) and \( S_i \) a schedule on a set of transactions \( T^i \), where \( T^0 \cap T^i = \emptyset \), \( 0 < i \leq n \). A schedule \( S_{ITS} \) is defined as follows:

\[ S_{ITS} = S_0 \big|_{i=1}^{n} (S_{transition} \mid S_i) \]

\[ T_1 = r_1(x)w_1(y)r_1(v)w_1(v); \quad T_2 = r_2(x)w_2(y); \quad T_3 = r_3(x)w_3(z); \quad T_4 = r_4(x)w_4(z) \]

\[ S = r_1(x)w_2(x)r_3(x)w_1(y)r_1(v)w_3(z)r_2(y)w_1(v); \quad S' = w_2(x)r_3(x)w_3(z)r_4(x)w_4(z)r_2(y) \]

\[ S'= r_1(x)w_2(x)r_3(x)w_1(y)r_1(v)w_3(z)r_4(x)w_4(z)r_2(y)w_1(v) \]

Fig. 3. Example of ordered join operation.
where $S_0$ is the initial schedule, before any change of the scheduler behavior, $S_i$, $0 < i < n$, is the schedule produced by ITS with a new behavior. The schedule $S_{transition}$ is the schedule generated during the transition phase and $i$ is the number of scheduler behavior changes. If the scheduler does not change its behavior, $S_{ITS} = S_0$.

Fig. 4 illustrates the notion of ITS schedule (see Definition 8). $S_{ITS}$ is defined by applying the ordered join operation (see Definition 7) on all schedules $S_i$ and $S_{transition}$, with $0 < i < n + 1$. In Fig. 4, each $S_{transition}$ is defined over transactions, which have operations synchronized by two different concurrency control protocols, one with conservative behavior and the other with aggressive. This occurs during the transition phase, when the scheduler (ITS) changes its behavior.

It is important to note that the schedules $S_i$ and $S_{transition}$ in Fig. 4 are in fact projections (see Definition 3) of $S_{ITS}$ over different sets of transactions. For example, $S_0$ is the projection of $S_{ITS}$ over the set of transactions submitted to ITS before any behavior change.

In order to implement a conservative or an aggressive behavior, the ITS uses well-known protocols, such as strict 2PL and timestamp ordering (TO) [28]. Although the ITS was implemented here with these two protocols, it is important to note that ITS may use any protocol based on serializability. The implementation codes of these protocols are stored in a component of the ITS called Protocol (see Fig. 1). Since the ITS dynamically changes concurrency control protocols, the challenge is to maintain the database in a consistent state during the transition phase.

The schedule produced during a transition phase involves the active transactions at the instant a behavior change notification arrives, called $T_{old}$, and the transactions submitted during the transition phase, called $T_{new}$. Thus, the set of transactions involved in the transition phase is $T_{transition} = T_{old} \cup T_{new}$. As already mentioned, the schedule defined for the transition period, called $S_{transition}$, is in fact the projection (see Definition 3) of $S_{ITS}$ over the set $T_{transition}$. While old transactions were scheduled according to a given protocol, the new transactions are scheduled by a different one. Therefore, operations in $S_{transition}$ are scheduled by two different protocols coexisting during the transition phase – a protocol $P_1$ used to schedule operations belonging to transactions in $T_{old}$, and the new protocol $P_2$, which schedules operations of transactions in $T_{new}$. Recall that $P_1$ and $P_2$ are based on serializability and present different behaviors, either aggressive or conservative.

In order to guarantee that $S_{transition}$ is correct, Axiom 1 (described next) must hold.

**Axiom 1.** The conflicting operations belonging to $T_{old}$ and $T_{new}$ should be synchronized in the same (serialization) order by both protocols (aggressive and conservative).

**Lemma 1.** Let $S_{transition}$ be a schedule produced by the ITS during the transition phase. Then $S_{transition}$ is a correct schedule.

**Proof.** During the transition phase, the ITS maintains two schedules – the schedule $S_{old}$ produced by the scheduler with the old protocol over $T_{old}$, and the schedule $S_{new}$ produced by a new protocol over $T_{new}$. The schedules $S_{old}$ and $S_{new}$ are correct since they are produced by concurrency control protocols based on serializability correctness criterion. According to Axiom 1, the ITS (during the transition phase) requires that the conflicting operations belonging to the sets $T_{old}$ and $T_{new}$ are serialized in the same order. Thus, $S_{transition}$ is correct. $\square$

**Theorem 1.** If $S_{ITS}$ is a schedule produced by ITS, then $S_{ITS}$ is correct.

**Proof.** Consider $S_{ITS}$ a schedule created by ITS, where

$$S_{ITS} = S_0 \parallel_{i=1}^n (S_{transition}, || S_i)$$

**Fig. 4.** ITS.
S₀ and Sᵢ are generated by protocols (with aggressive or conservative behavior) that use protocols based on already proven correctness criteria (e.g. serializability). The schedule S_transition is correct (Lemma 1). Therefore, the serialization graphs for the schedules S₀, S_transition, and Sᵢ do not have cycles. The schedules generated by ordered join operations in SSTS, S || S_transition and S transition || Sᵢ are correct, that is, they do not introduce cycles in the SG(SSTS), because:

1. the schedule order of the common operations on the schedules involved in an ordered joint operation is preserved (item (iv) of Definition 7), and
2. S₀ and Sᵢ are defined over two sets of transactions T₀ and Tᵢ, where T₀ ∩ Tᵢ = Ø, 0 < i ≤ n (Definition 8).

Since SG(SSTS) does not have any cycle, SSTS is correct. □

In order to describe how the ITS correctly schedules operations of concurrent transactions, let Tᵢ (0 < i ≤ n) and Tⱼ (0 < j ≤ m), i ≠ j, be transactions, where Tᵢ ∈ Told and Tⱼ ∈ Tnew. The ITS schedules operations according to the following rules:

- **R1**: ∀ pᵢ ∈ OP(Tᵢ), pᵢ must be scheduled by the protocol that was in use before the transition phase, and ∀ qⱼ ∈ OP(Tⱼ), qⱼ must be scheduled by the new protocol;
- **R2**: ∀ pᵢ ∈ OP(Tᵢ), ∀ qⱼ ∈ OP(Tⱼ), where pᵢ conflicts with qⱼ, pᵢ and qⱼ must be scheduled during the transition period according to the rules of the new and old protocols;
- **R3**: Operations of Tᵢ have execution priority over operations of Tⱼ. That means, operations of Tᵢ that conflict with operations of Tⱼ may induce the abort of Tⱼ.

It is important to note that rule R1 guarantees that each transaction is scheduled by the protocol chosen by the Analyzer. In turn, rule R2 ensures that during the transition period (recall that during this period the scheduler maintains two behaviors, i.e., conservative and aggressive) the serialization graph for S_transition remains acyclic. For that, the conflicting operations should have the same serialization order; independently of the type of behavior they have been scheduled. In other words, R2 guarantees Axiom 1 and Lemma 1. Observe that if Axiom 1 and Lemma 1 hold, then Theorem 1 holds as well. Finally, rule R3 defines a strategy to guarantee R2, since it may be necessary for the aggressive protocol to delay conflicting operations of Tⱼ, or the conservative protocol may need to reject conflicting operations of Tⱼ, aborting the transaction.

### 3.3. ITS implementing semantic serializability

In the previous section, the ITS using classical (syntactic) serializability as correctness criterion is described. This model has been quite appropriate for conventional database applications. However, the nature and requirements of transaction processing in dynamically configurable environments (e.g., an MDBC) are quite different from those in conventional applications. For instance, in an MDBC two different types of transactions may be executed, mobile and local transactions. A local transaction is submitted directly to a mobile database on the same host. A mobile transaction, denoted Mₑ, consists of a set of sub-sequences (SUB₁ₑ, SUB₂ₑ, SUB₃ₑ, …, SUBᵢₑ) of database operations, where each SUBₑ is executed at a mobile database MDBₑ as an ordinary (local) transaction.

Although mobile and local transactions coexist, in an MDBC there is no central component that has information on the existence and execution order of local transactions due to local autonomy requirements. On the other hand, syntactic serializability requires knowledge of the execution order of all active transactions. Therefore, the conventional concurrency control model is unsuited for processing transactions in dynamically configurable environments, such as an MDBC. For that reason, a correctness criterion, denoted Semantic Serializability (SₛSᵢ), was introduced into the ITS for synchronizing transactions in mobile database environments. Since Semantic Serializability is more permissive than classical syntactical serializability, it provides a higher degree of inter-transaction parallelism of mobile transactions by relaxing the atomicity property [5,6].

The SₛSᵢ model is based on the use of semantic knowledge to relax the notion of absolute transaction atomicity [4]. The key idea of that transaction model is to “see” an MDBC as a collection of disjoint sets of objects, each of which represents a single mobile database. Those disjoint sets of objects are called semantic units (SU). It is reasonable to assume that the result of an update action executed by a mobile transaction on an object belonging to a particular semantic unit does not depend on the values of objects belonging to other semantic units that were previously read by the same transaction. In other words, there are no update dependencies between objects of different semantic units. With this approach a transaction may consist of several atomic units. An atomic unit (Definition 9) is the set of operations of the transaction over a semantic unit [4].

**Definition 9 (Atomic unit – AU).** Let T be a transaction on database DB = ∪ᵢ₌₁ⁿSUᵢ, where SUᵢ, 0 < i ≤ n are semantic units defined for DB. An atomic unit ∏ᵢ₌₁ⁿSUᵢ(T) for T consists of a sequence of read and write operations executed by T on objects of SUᵢ, where:

∀ p, q ∈ OP_SUᵢ(T), p < ∏ᵢ₌₁ⁿSUᵢ(T), q ⇔ p < r q

M. Holanda et al. / Data & Knowledge Engineering 66 (2008) 223–242
In the S\(_S\) transaction model, a schedule \(S\) is correct iff \(S\) is semantically serializable. The semantic serialization graph is used to check if a schedule is semantically serializable. The graph of a semantic serializable schedule \(S\) does not contain any cycles over edges with the same label of the form \(SU_i\). The Semantic Serialization Graph for \(S\) is a directed graph \(SSG(S) = (N,E)\) in which \(N = T, T\) is a set of transactions, and \(E\) represents the set of labeled edges \(T_i \xrightarrow{SU_i} T_j\), where \(T_i, T_j \in N\), and there are two operations, which are in conflict, \(p \in OP(T_i), q \in OP(T_j), p \preceq q\), on an object of the \(SU_i\). In [4], some concurrency control protocols with exclusive aggressive or conservative behavior based on \(S_S\) correctness criterion are presented and the \(S_S\) model is discussed more thoroughly.

By definition, a schedule \(S^{ITS}\) is composed of the following schedules (see Definition 8):

(i) the schedules produced by conservative and aggressive protocols, and
(ii) the schedule in the transition phase.

Definition 8 is always valid regardless of which correctness criterion the ITS is implementing. Once more, the challenge of the ITS is to maintain the database in a consistent state during the transition phase. Recall that the transition phase begins with a notification of behavior change. For ITS implementing \(S_S\) correctness criterion, the transition phase ends when the atomic units, which were active (Definition 10) at the time of notification, change their states to completed atomic units (Definition 11).

Definition 10 (Active atomic unit). An atomic unit is active if:

(i) \(o_n\) is its last operation and has not yet been successfully executed.

Definition 11 (Completed atomic unit). An atomic unit is said to be completed if:

(i) \(o_n\) is the its last operation and has been successfully executed, or
(ii) the atomic unit has been aborted.

Using the atomic unit concept, the same transaction \(T\) may be scheduled by the ITS using concurrency control protocols with different behaviors. Fig. 5 illustrates an example of a transaction \(T\) composed of three atomic units, where each atomic unit was scheduled with a different behavior. Transaction \(T\) began being scheduled with conservative behavior, went through a transition period, where the schedule behavior is hybrid, and ended with an aggressive concurrency control protocol.

Accordingly, the process of producing correct schedules is even more complex when ITS is using the \(S_S\) model than when it is using classical serializability. When ITS schedules transactions using classical serializability, each schedule (see Fig. 4) is generated by only one concurrency control protocol (either aggressive or conservative). On the other hand, when there is a behavior change in the ITS using \(S_S\) correctness criterion, each schedule may be produced by different protocols. In order to illustrate this assertion, consider Fig. 6, where \(AU_1, AU_2\) and \(AU_3\) are atomic units of a given transaction \(T\). In turn, \(T\) belongs to the set of transactions, over which the schedule \(S_0\) is defined. The atomic units \(AU_1\) and \(AU_2\) were active atomic units (Definition 10), when the transition phase started, while \(AU_3\) was a new atomic unit. Thus, \(S_0\) was produced by protocols with different behaviors, since \(S_0\) has old atomic units (initiated before the behavior change notification, \(AU_1\) and \(AU_2\) ) and new atomic units (initiated after the behavior change notification, \(AU_3\)).

\[
T = r(a) r(k) w(b) r(x) w(j) w(z)
\]

\[
T = \left\{ \prod_{OP_1(T)}(T), \prod_{OP_2(T)}(T), \prod_{OP_3(T)}(T) \right\}, \text{where :}
\]

\[
\prod_{OP_1(T)}(T) = r(a) w(b) \quad \prod_{OP_2(T)}(T) = r(k) w(j) \quad \prod_{OP_3(T)}(T) = r(x) w(z)
\]

Fig. 5. Example of a transaction \(T\) with three atomic units.
Therefore, in order to make ITS able to implement $S\_S$ correctness criterion, the notion of schedule should be redefined, since a schedule may be generated by different concurrency control protocols, each of which presents different behaviors (conservative or aggressive).

**Definition 12** ($S\_S$ Schedule). Let $S$ be a schedule defined over the set of transactions $T$, $S_{\text{completed}}$ be the projection of the transactions belonging to $T$, which has finished their execution before the behavior change notification, and $S_{\text{active}}$ be the projection of the transactions of $T$ that were active at the moment of the behavior change notification. The schedule $S$ is defined as $S = S_{\text{completed}}||S_{\text{active}}$.

Fig. 6 illustrates the notion of an $S\_S$ schedule (Definition 12). The schedule $S$ represents the execution of the set of the transactions $T$ that initiated their execution with protocol P1. This schedule involves all the transactions that were completed before the behavior change notification ($S_{\text{completed}}$), and all the transactions that were active ($S_{\text{active}}$) at the moment of the behavior change notification.

In the following, it is proven that a schedule $S$ produced by ITS using semantic serializability is correct, i.e., $S$ is semantically serializable ($S \in S\_S$).

**Lemma 2.** If $S$ is a schedule produced by ITS using the $S\_S$ correctness criterion and in accordance with Definition 12, then $S$ is correct, i.e., $S$ is semantically serializable ($S \in S\_S$).

**Proof.** Let $S$ be a scheduler produced by ITS using $S\_S$ as correctness criterion.

Case 1. The schedule is produced when there is no behavior change. This is trivial, since $S$ is produced by just one concurrency control protocol.

Case 2. Schedule $S$ is generated during a behavior change.

Thus, $S = S_{\text{completed}}||S_{\text{active}}$ by Definition 12. The schedule $S$ is correct iff its semantic serializability graph does not contain any cycle (this is proven in [4]). By definition, the projection $S_{\text{completed}}$ (see Definition 12) is generated by protocols (with either aggressive or conservative behavior) based on $S\_S$ correctness criteria (e.g., Sel). Thus, such protocols do not synchronize operations which introduce a cycle in the graph $SSG(S)$ [4]. The projection $S_{\text{active}}$ does not introduce cycles in the graph $SSG(S)$, because: (i) the atomic units initiated before the behavior change notification and the new atomic units, initiated after the behavior change, are scheduled by protocols based on $S\_S$ correctness criterion, whose correctness are proven in [4]; (ii) ITS (during the transition period in Fig. 6) requires that the conflicting operations belonging to active atomic units and new atomic units be synchronized in the same (semantic serialization) order for each semantic unit. Finally, the operation $S = S_{\text{completed}}||S_{\text{active}}$ does not produce any cycle, since, by Definition 7, the order of the common operations on the schedules involved with the ordered join operation is preserved (item (iv) of the Definition 7). Then the graph $SSG(S)$ does not have cycles, therefore $S$ is correct.

The schedule produced during the transition period by ITS implementing the $S\_S$ model, denoted $S_{\text{transition}}$, involves operations of two different sets of atomic units: the set of the active atomic units at the instant a behavior change notification arrives, called $AU_{\text{old}}$, and (ii) the set of new atomic units, which began during the transition period, with the new behavior, called $AU_{\text{new}}$. Furthermore, the following relationships are valid: $AU_{\text{old}} \subseteq T_{\text{old}}$ (recall that $T_{\text{old}}$ represents the set of transactions initiated after the behavior change notification) and $AU_{\text{new}} \subseteq T_{\text{old}} \cup T_{\text{new}}$, where $T_{\text{new}}$ represents the set of transactions initiated after the behavior change notification. In Fig. 5, the transaction $T$ has atomic units belonging to the sets $AU_{\text{old}}$ and $AU_{\text{new}}$. $S_{\text{transition}}$ (see Definition 8) represents the projection of $S_{\text{ITS}}$ over the set $T_{\text{transition}}$, where $T_{\text{transition}} = T_{\text{old}} \cup T_{\text{new}}$. Therefore, operations in $S_{\text{transition}}$ are scheduled by two different protocols coexisting during the transition phase – a protocol P1 used to schedule operations belonging to active atomic units in $AU_{\text{old}}$, and the new protocol P2, which schedules operations of atomic units in $AU_{\text{new}}$. In order to guarantee that $S_{\text{transition}}$ is correct, Axiom 2 should hold. □
Axiom 2. The conflicting operations belonging to \( AU_{\text{old}} \) and \( AU_{\text{new}} \) should be synchronized in the same (semantic serialization) order by both protocols (aggressive and conservative) over each semantic unit.

Lemma 3. Let \( S_{\text{transition}} \) be a schedule produced by the ITS using \( S,S \) during the transition phase. Then \( S_{\text{transition}} \) is semantically serializable (i.e., \( S_{\text{transition}} \) is correct).

Proof. During the transition phase, the ITS maintains two schedules: the schedule \( S_{\text{old}} \) produced by the old protocol over the set of the active atomic units (\( AU_{\text{old}} \)) at the moment of the behavior change notification; and the schedule \( S_{\text{new}} \) produced by new protocol over the atomic units initiated after the behavior change notification (\( AU_{\text{new}} \)). The schedules, \( S_{\text{old}} \) and \( S_{\text{new}} \), are correct, since they are produced by concurrency control protocols based on semantic serializability correctness criterion [4].

According to Axiom 2, the ITS (during the transition phase) requires that the conflicting operations belonging to the sets \( T_{\text{old}} \) and \( T_{\text{new}} \) of atomic units are synchronized in the same (semantic serialization) order for each semantic unit. Consequently, \( S_{\text{transition}} \) is correct.

Theorem 2 proves that the schedules produced by ITS using \( S,S \) correctness criterion are correct.

Theorem 2. If \( S^{\text{ITS}} \) is a schedule produced by ITS using \( S,S \), then \( S^{\text{ITS}} \) is correct.

Proof. Consider \( S^{\text{ITS}} \) a schedule created by ITS, where \( S^{\text{ITS}} = S_0 \|_{i=1}^{n} (S_{\text{transition}}, \|S_i\|) \). \( S_0 \) and \( S_i \) are correct (by Lemma 2). The schedule \( S_{\text{transition}} \) is correct (Lemma 3). Therefore, the semantic serialization graphs for the schedules \( S_0, S_{\text{transition}} \) and \( S_i \) do not have cycles over same label (semantic unit). The schedules generated by ordered join operations in \( S^{\text{ITS}}, S_{\text{transition}} \) and \( S_i \) are correct in that they do not introduce cycles with the same label in the SSG(\( S^{\text{ITS}} \)). This is because (i) the order of schedule) of the common operations on the schedules involved in an ordered join operation is preserved over each semantic unit, and (ii) \( S_0 \) and \( S_i \) are defined over two sets of transactions \( T^{\text{old}} \) and \( T^{\text{new}} \), where \( T^{\text{old}} \cap T^{\text{new}} = \emptyset \), \( 0 < i < n \) (Definition 8).

Since SSG(\( S^{\text{ITS}} \)) does not have cycles over same label \( U_0, S^{\text{ITS}} \) with \( S,S \) is correct. □

3.4. Implementation issues

This section describes how the ITS produces schedules in a dynamically configurable environment by using either classical serializability or semantic serializability (see Sections 3.2 and 3.3). Without lost of generality, the concept of MDBC to characterize a dynamic environment is used.

Users interact with mobile databases in an MDB by invoking transactions. A transaction represents a sequence of operations on database objects. There are two types of transactions in an MDB environment: local transactions and mobile transactions. A local transaction is submitted directly to a mobile database on the same host. A mobile transaction, denoted \( M_i \), consists of a set of subsequences \( \{SUB_{i,1},SUB_{i,2},SUB_{i,3},...,SUB_{i,n}\} \) of database operations, where each \( SUB_{i,k} \) is executed at a mobile database \( MDB_k \) as an ordinary (local) transaction. In other words, a mobile transaction \( M_i \) is defined as follows: \( M_i = \{SUB_{i,1},SUB_{i,2},SUB_{i,3},...,SUB_{i,n}\} \), where each \( SUB_{i,k} \) is executed by mobile database \( MDB_k \). The notion of mobile transaction models distributed transactions, which are composed of mobile subsequences of database operations. One of the main problems of concurrency control in an MDB is that each local DBMS has the autonomy to manage its own transactions.

In an MDB, a local schedule models the execution of several interleaved operations belonging to local and mobile transactions in a given mobile unit. A global schedule \( S \) (Definition 10) represents the execution of all operations of the mobile and local transactions in the MDB [5,6].

Definition 10 (Global schedule). A global schedule \( S \) is a schedule resulting from the concurrent execution of all the transactions, \( T = M \cup L \) (local and mobile transactions), at all the local databases. For any mobile unit, \( MU_k \), a projection of \( S \) on a set of mobile and local transactions executing at mobile host \( MU_k \) is the local schedule \( S_i \) at mobile unit \( MU_k \).

Since each mobile unit is autonomous on the execution of the local and mobile transactions, another component is necessary to guarantee the correct global schedule. In this context the ITS is responsible for scheduling the mobile transactions.

The problem of maintaining correct global schedules in an MDB is complicated by the presence of local transactions that are invisible at the global level (by ITS). These invisible local transactions can cause indirect conflicts among mobile transactions that do not conflict based on global information [17]. Furthermore, ensuring that each local schedule is correct does not ensure that the global schedule is also correct [14].

In order to guarantee correct global schedules using the classical serializability correctness criterion, the necessary and sufficient conditions are [11,14,17,29]: (i) every local schedule is serializable; (ii) there is a total order \( O \) over mobile transactions, such that at each \( MU \) the serialization order of mobile subtransactions is consistent with \( O \). Since all existing database systems implement serializability, we assume that local schedules are always serializable. For the second condition, the ITS uses the implicit ticket method (ITM) [11].

On the other hand, to produce correct global schedules according to the \( S,S \) transaction model, the ITS should ensure the following conditions [2,6]: (i) every local schedule is serializable, and (ii) the projection of the global schedule (see Definition 10) on the set of mobile transactions is semantically serializable.
4. Experimentation results

In order to evaluate the performance of the proposed self-adaptable scheduler (ITS), its operation was compared with schedulers implementing a static behavior (aggressive or conservative). First, the performance of the ITS with classical serializability transaction model was evaluated. So, the ITS was compared with schedulers implementing either strict 2PL or timestamp ordering (TO) protocols. Thereafter, the performance of the ITS implementing protocols based on semantic serializability model was evaluated. The performance of these protocols was evaluated based on the following metrics:

- **Lock waiting rate**: This metric indicates the percentage of the number of operations waiting for a lock to be released. When this rate is high, it means that the throughput is low.
- **Abort rate**: This metric indicates the number of aborts that occurred for a set of concurrent transactions.
- **Conflicting operation rate**: This metric measures the percentage of the conflicting operations in relation to the total number of operations.

4.1. Simulation environment

In order to simulate the execution of the ITS in a dynamically configurable environment, a mobile database community (MDBC) was used, formed by $n$ mobile units interconnected by a wireless network. Moreover, any mobile unit can leave the network and new mobile units can join the MDBC. The simulation environment has three components: the ITS, the TransactionSimulator and the MDBCsimulator, which are described as follows.

The MDBCsimulator component simulates a dynamically configurable environment. In fact, that component simulates an MDBC whose topology may change dynamically due to the mobility of mobile databases (residing in mobile hosts). By means of that component, it is possible to define:

- the initial number of the mobile units, and the amount of available data items in each mobile unit for the MDBC;
- if and when (in milliseconds) a mobile unit belonging to the MDBC should leave the MDBC, and
- when new mobile units join the MDBC.

The transaction simulator, TransactionSimulator, is a software component that creates mobile transactions, and submits them to the ITS. The operations of the mobile transactions are created randomly over data items of the global scheme. With the TransactionSimulator, the user may configure the number of transactions, the amount of data items involved in the transactions, the percentage of read and write operations in each transaction, and the dispatch interval (in milliseconds) of each operation.

The ITS implementation is composed of two main modules: the analyzer, implemented with the JFuzzy API [13] and the scheduler. In order to initiate the ITS’ operation, the user must: configure the fuzzy variable values used by the analyzer, and choose the initial scheduler behavior.

For the scheduler module, the protocols TO (aggressive) and 2PL (conservative) were implemented. Recall that those protocols are based on classical serializability. For semantic serializability-based concurrency control protocols, the ITS’ scheduler module implements the protocols SeTO (aggressive) and SeL (conservative).

The configuration of each component in the simulated mobile environment is made by means of a graphic interface. With this simulator, the user can configure several test scenarios. All components were developed using Java language.

4.2. Analysis of results

In order to use the ITS, the administrator should configure the fuzzy variable values used by the Analyzer to select the scheduler behavior (see Section 2.1). For all the tests presented herein, the following values were assigned: $atr$ and $lwr$ were considered low when between 0 and 20, medium between 40 and 60, and high when between 80 and 100; and $cor$ is considered low when its value is between 0 and 20, medium when it varies from 40 to 60, and high when it ranges from 80 to 100. These values were configured by a specialist using the following strategy. The specialist ran many tests with aggressive or conservative schedulers for defining these values. These values were coherent with the rules defined for the Analyzer.

Different test scenarios were configured. In order to demonstrate that the ITS is able to identify changes and automatically decide on the most appropriate behavior, scenarios were simulated in which initially all transactions were composed of read operations. After a period of time, a new mobile database joins the MDBC, which submits mobile transactions composed of write operations.

The first two test scenarios were used to evaluate the ITS implementing syntactical serializability-based concurrency protocols, namely strict 2PL and TO. In the first scenario, the simulated environment (Fig. 7) is initially composed of an MDBC with 3 mobile units, MU$_1$, MU$_2$ and MU$_3$. After five seconds a new mobile unit, MU$_4$ joins the MDBC. Each mobile unit has an identifier. The MU$_1$ and MU$_2$ have a transaction simulator and a database composed of 20 available data items for the MDBC. The MU$_3$ has only one database. The MU$_4$ does not create mobile transactions. The mobile unit MU$_4$ has the transaction...
simulator, but does not have a database for the MDBC. The transactions generated by MU₁ and MU₂ are composed of read operations only and are executed over the global scheme of the MDBC. The mobile transactions generated by MU₄ are composed of 80% read operations (and 20% write operations). The write operation is always executed over one data item of the global scheme. The initial scheduler behavior is conservative.

In order to evaluate the ITS’ performance, in the first scenario, five sets of tests were run in which the ITS synchronized operations of 100, 200, 300, 400 and 500 mobile transactions, respectively. In the tests with sets of 100 and 200 mobile transactions, the results obtained (Figs. 8–10) were derived from the transactions generated by MU₁ and MU₂ (50 and 100 transactions respectively). In the tests with 300, 400 and 500 transactions, the transaction simulators of the MU₁ and MU₂ generated 100 mobile transactions each, and the MU₄ generated 100, 200 and 300 mobile transactions, respectively. The tests were done in the following manner: for each set of tests, three different schedulers were used. First, the ITS was used to synchronize the operations of the mobile transactions during the entire period. The conservative behavior of ITS was implemented by means of the strict 2PL protocol, while the aggressive behavior was implemented by TO protocol. At a second given moment, a scheduler implementing only strict 2PL was used. Finally, a scheduler-based exclusively on the TO protocol was used.

In Figs. 8–10, one can observe that the schedulers implementing 2PL and TO protocols behaved similarly to ITS up to 200 transactions (tests done without MU₄) regarding lock waiting rate (Fig. 8) and aborted transaction rate (Fig. 9) metrics. Since all transactions for the scenario depicted in Figs. 8–10 are composed of only read operations, there will be no conflicting operations, consequently there will be neither operations in queue, nor aborted transactions.
After the new mobile unit, MU4, entered the MDBC, which occurs 5 s after the initial MDBC is formed, the ITS demonstrated better performance than the 2PL in relation to the lock waiting rate (Fig. 8), maintaining the aborted transaction rate low (Fig. 9). In this scenario, the ITS initially began with conservative behavior (2PL) and changed to aggressive behavior (TO). The protocol that presented the worst performance in relation to lock waiting rate was the 2PL, because the mobile transactions generated by MU4 have one write operation executed over only one data item (a type of hot spot), and there is a high number of operations waiting for the lock in this data item to be released. Note that in these tests the conflicting operation rate (Fig. 10) was low, lower than 20%, and the aborted transaction rate was between 0% and 20% (Fig. 9), during the entire test period. The ITS' Analyzer used Rule 1 (\(\text{IF atr } = L \text{ AND cor } = L \text{ AND lwr } = H \text{ THEN sb } = A\)) to define that the aggressive behavior was the most appropriate.

In the second scenario, the MDBC is formed by the same mobile units described in the first scenario. The mobile unit MU4 also enters the MDBC after 5 s. The differences are: the mobile transactions generated by MU4 are composed of 20% read operations and 80% write operations. Those operations are executed over a set of 20 data items. In order to evaluate the ITS performance, the same five sets of tests were executed. Moreover, the ITS’ initial behavior was set to aggressive.

Fig. 11 shows that the aborted transaction rate of the three schedulers (ITS, 2PL and TO) were the same (\(tta = 0\)) up to 200 transactions, since all operations are read operations. After MU4 entered, the TO scheduler increased the aborted transaction rate, because the mobile transactions generated by MU4 have many write operations, generating many conflicting operations (Fig. 13) and aborted transactions (Fig. 11). In these tests the ITS began with aggressive behavior and changed to conservative behavior. The ITS’ Analyzer used Rule 7 (\(\text{IF atr } = M \text{ THEN sb } = C\)) to recognize that the conservative behavior was the most appropriate.

Looking more closely at Figs. 8, 9, 11 and 12 one can observe that the ITS is able to find a good compromise between the abort rate and lock waiting rate compared to schedulers implementing a fixed concurrency control protocol (in our tests strict 2PL and TO). Therefore, such results prove our claim that an adaptive scheduler is quite appropriate for controlling concurrency in dynamically configurable environments, such as an MDBC.

When the ITS implements a correctness criterion more permissive than syntactic serializability, this claim holds as well, as shown in the following graphics. The same two scenarios (first and second scenario) were used to evaluate the use of the ITS in a dynamic environment using semantic serializability [4] as correctness criterion. These tests are presented as follows.

In the third scenario, five sets of tests were run in which the ITS synchronized operations of 100, 200, 300, 400 and 500 mobile transactions, respectively. In the tests with sets of 100 and 200 mobile transactions, the results obtained (Figs. 14–16)
were derived from the transactions generated by MU1 and MU2 (50 and 100 transactions, respectively). For each set of tests, three different schedulers were used, all using the semantic serializability correctness criterion, the first implementing the SeL (conservative), the second implementing SeTO (aggressive) and finally the ITS. The schedulers implementing SeL and SeTO protocols behaved similarly to the ITS when the tests were executed without the mobile unit MU4, because MU1 and MU2 submit transactions composed of read operations only. During this period (before the arrival of the MU4) the ITS maintained the conservative behavior. The Analyzer used Rule 3 (IF atr = L AND lwr = L AND isb = C THEN sb = C) for identifying that the ITS Scheduler should continue with conservative behavior.

After the arrival of the new mobile unit, MU4, into the MDBC, which occurs 5 s after the MDBC is formed, the ITS presented more efficient behavior than the scheduler implementing SeL. The ITS, with an initial conservative behavior, changed it to aggressive. Similar to the case of classical serializability, the scheduler that presented the worst performance in relation to lock waiting rate (Fig. 14) was the one implementing SeL (with exclusive conservative behavior), because the mobile

![Conflicting operation rate](image1)

**Fig. 11.** Conflicting operation rate for the first test scenario.

![Aborted transaction rate](image2)

**Fig. 12.** Abort rate for the second test scenario.

![Lock waiting rate](image3)

**Fig. 13.** Locking waiting rate for the second test scenario.
transactions generated by MU4 have one write operation executed over one data item only, and there is a high number of operations waiting for the unlocking of this data item. The difference between the ITS with semantic serializability and the ITS with classical serializability is that with classical serializability this rate is based on the entire transaction, and with semantic serializability this rate is based on operations of the atomic units. The Analyzer also used Rule 1 (IF \[ atr = L \text{ AND } cor = L \text{ AND } lwr = H \] THEN \( sb = A \)) to determine that the most appropriate behavior is aggressive.

In the fourth scenario, the ITS with semantic serializability correctness criterion was applied where the MDBC was composed of the same mobile units, MU1, MU2, MU3 and MU4. The mobile unit MU4 also enters the MDBC after 5 s. The mobile transactions generated by MU4 are composed of 20% read operations and 80% write operations. The ITS’ initial behavior was aggressive. In order to evaluate the ITS, the tests were executed with three schedulers: SeL, SeTO and ITS.

In this scenario, after the MU4 entered, the SeTO scheduler increased the aborted transaction rate (Fig. 17). MU4 submits transactions with many write operations, generating many conflicting operations (Fig. 18). In these tests the ITS behaved more efficiently than the SeTO w.r.t. abort rate. The ITS dynamically changed its behavior from aggressive to conservative. The Analyzer used Rule 7 (Rule 7: IF \( atr = M \) THEN \( sb = C \)) (see Fig. 19).
5. Related work

In the literature, there are some proposals for concurrency control algorithms with aggressive and conservative behavior. There are even algorithms that integrate aggressive and conservative characteristics with the goal of achieving better performance from the scheduler. However, none of these proposals supports the notion of a self-adaptable scheduler, with the ability of dynamically changing its behavior to adapt itself to the characteristics of the computing environment. Furthermore, none of these proposals uses a fuzzy expert system to facilitate the decision of which behavior is the most appropriate for a scheduler at any given moment. Finally, the manner in which the ITS produces correct schedules and ensures, thus, database consistency (see Section 3) is also innovative. Some of these aspects are discussed as follows.

Bernstein and Goodman presented in [28] algorithms that integrate characteristics of 2PL and TO. Their idea was to divide the synchronization process into two groups: rw synchronization (rw – read and write operations) and rwr synchronization (read–write–read operations). An interface between a 2PL and a TO algorithm was created to guarantee correct schedules,
since both protocols can be used for rw and ww synchronization. This interface uses locked points to induce timestamps. A disadvantage of this approach is that either the set of write operations is pre-declared or a write operation wait queue is necessary. Moreover, this approach, unlike the ITS, does not present any scheduler behavior change.

Bhargava [10] defined algorithmic adaptability as the set of techniques for dynamically changing from the execution of one algorithm to a different one. In his proposal, adaptability can take place in one of three ways: temporal adaptability, which refers to changes over time (this method generally has a brief conversion period); per-transaction, which consists of methods that allow each transaction to choose its own algorithm, different transactions running at the same time may run different algorithms; spatial adaptability is a variant of per-transaction adaptability in which transactions choose the algorithm based on properties of the data items they access. The method presented by Bhagava, which comes closer to the one presented by ITS, is the suffix-sufficient temporal adaptability. The advantage of the ITS is that in Bhargava’s proposal it is necessary to define a termination function so that the scheduler can change to the new algorithm, whereas in the ITS this function is not necessary.

In [1], Akintola et al. propose a hybrid technique called optimistic locking architecture, which provides a locking mechanism when there are many hot spots (data items accessed concurrently by many conflicting operations) in the system, and an optimistic mechanism when most of the transactions are accessing data which do not represent hot spots (i.e., data accessed by non-conflicting operations). In order to define which mechanism should be applied at a given moment, the approach proposed in [1] uses a data structure called lock buffer. According to the authors, by means of such a structure it is possible to identify the existence of hot spots, which represent the only variable used by the scheduler to make a decision about changing its behavior. By doing this, the scheduler may react too late. To illustrate this claim, consider that the scheduler is using an optimistic technique at a given moment, when suddenly several transactions are submitted. Suppose that those transactions are accessing a small set of data items with conflicting operations, producing, thus, a hot spot. Since the scheduler is using an optimistic technique and there is a hot spot, it is highly likely that the aborted transaction rate increases for a period of time. This scenario does not occur with the ITS, since it uses an expert system (with several rules) to make a decision about changing its behavior. The ITS maintains the aborted transaction rate at the lowest possible level.

In [8], Helal et al. present an adaptable transaction scheduling technique that is applicable to high contention environments, being used in conjunction with any lock-based concurrency control algorithm. The goal of this proposal is to use transaction priority scheduling. For that reason, the scheduler (called adaptor server) should use the following parameters to determine the priority of each transaction: transaction size, number of operations that have been executed in a transaction, transaction’s abort count, transaction’s conflict count, transaction’s restart count, transaction’s hot spot access percentage and transaction’s read/write ratio. These parameters are combined in the non-linear aging function, which provides some measure of the duration of time that a transaction has waited in the CC queue. Differently from the ITS, this proposal does not change the scheduler behavior, but rather only uses the transaction’s priority in the conservative protocol (based on block).

Besides having an adaptive behavior, the ITS can implement different correctness criteria, such as classical (syntactic) serializability and semantic serializability (SeS). Since semantic serializability is more permissive than classical syntactical serializability, ITS is able to provide a higher degree of inter-transaction parallelism of transactions for advanced database applications, such as computer-aided design and software engineering (CAD and CASE), geographic information systems (GIS) and workflow management systems (WFMS). Such applications consist of long-living and (possibly) interactive transactions.

Most of the transaction models presented in the literature related to mobile computing assume that mobile units access databases located in fixed hosts [14,22,23,27]. On the other hand, approaches for transaction management related to dynamically configurable environments, such as an MDBC, are based on the use of concurrency control protocols with exclusive aggressive or conservative behaviors, i.e. they are based on schedulers with static behavior. Two of such protocols are discussed in the following.

SESAMO [5] is a concurrency control mechanism based on semantic serializability [4]. According to SESAMO mechanism, the mobile transaction concurrency control is distributed among several independent mobile multidatabase transaction schedulers (called MOTSs), each one producing an independent schedule called Partial Global Semantically Serializable Schedule (PGS3). The activities of several MOTSs can be organized as a single schedule called Single Global Semantically Serializable Schedule (SGS3) resulted from the union of all the PGS3 produced by the corresponding MOTSs. This proposal implements an aggressive behavior.

Gruenwald and Banik in [20] present a description of a model for transaction management in MANETs. Depending on the communication capacity, memory size and energy limitations, the mobile units are classified in two groups: Small Mobile Host (SMH) and Large Mobile Host (LMH). When all sub-transactions of a global transaction have been completed, a global partial serialization graph is executed, to verify the global atomicity and isolation properties. This proposal presents only an aggressive behavior, as well. Recall that approaches implementing concurrency control protocols with aggressive behavior present high aborted transaction rates when executed over hot spots.

Table 1 summarizes the proposals described in this section regarding the following criteria: the ability of presenting hybrid behavior, suitability of being used in dynamically configurable environments, and the support for protocols based on conventional serializability and/or based on a correctness criterion which relaxes syntactic serializability. According to Table 1, only ITS presents hybrid behavior, can be used in dynamically configurable environments, and implements different correctness criteria (i.e., serializability and semantic serializability).
Table 1
Summary of the models’ characteristics

<table>
<thead>
<tr>
<th>Hybrid behavior</th>
<th>Dynamic environment</th>
<th>Serializability</th>
<th>Semantic knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernstein and Goodman [28]</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Bhargava [10]</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Helal [8]</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Akintola [1]</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SESAMO [5]</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gruenwald and Banik [20]</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ITS</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

6. Conclusion

This paper presented the ITS – a scheduler that adapts its behavior dynamically and automatically (without human interference) according to the computing environment characteristics (e.g., aborted transaction rate, conflicting operation rate and lock waiting rate). Therefore, the ITS is a self-adaptable scheduler that reacts to changes in computing environments. Such a feature is critical for a scheduler running in dynamically configurable environments, whose topology changes dynamically. The ITS was developed to synchronize transactions by using syntactical serializability and semantic serializability as correctness criteria. The ITS always produces correct schedules, and the formal proofs of the correctness are presented in Section 3.

In order to evaluate the viability of the ITS using serializability, two protocols, 2PL (conservative behavior) and TO (aggressive behavior), were implemented in the core of ITS. In order to evaluate the ITS using semantic serializability, two protocols were implemented, Sel (conservative) and SeTO (aggressive). However, it is important to note that the ITS may be used with any protocol based on serializability or semantic serializability. The ITS was compared to schedulers with fixed behavior (aggressive or conservative).

The tests presented in Section 4 simulated the use of the ITS for synchronizing transactions in an MDBC — a dynamically configurable environment. The analysis of the simulation results shows that the ITS (with hybrid behavior) is quite efficient for controlling concurrency in dynamic environments, such as an MDBC, since it is able to find a good compromise between the abort rate and lock waiting rate compared to schedulers implementing static behavior, i.e., fixed concurrency control protocols.

The results obtained demonstrate the viability of employing the ITS in a dynamic environment, such as an MDBC. Nonetheless, other functions are being investigated, such as: implementation of the ITS with correctness criteria other than serializability and serializability semantic; distributed processing of the ITS, which is an important feature in a distributed environment; and an Analyzer (ES) specification for mobile environments, including the following variables: number of mobile units, disconnect rate and available rate.

References


Maristela Holanda received the Electronic Engineering Diploma from the Federal University of Rio Grande do Norte – UFRN, Brazil, in 1996. She received the MSc degree from University of Brasilia, Brazil, in 1999. She received the PhD degree from the University of Rio Grande do Norte – UFRN, Brazil, in 2007. Since 2000 she has been working for the Catholic University of Brasilia – UCB, at the Department of Computer Science. Her current research interests include and mobile databases and transaction concurrency control systems.

Angelo Brayner received the MSc degree in Computer Science from the State University of Campinas (UNICAMP), Brazil, in 1994. He received the PhD degree from the University of Kaiserslautern, Germany. He has been with the University of Fortaleza, Brazil, since 2001 as full professor and leader of the CEARA (advancEd dAtabase Research) research group. His current research interests include query processing in wireless sensor networks and mobile databases and high performance transaction systems.

Sergio Vianna Fialho was born in Rio de Janeiro, Brazil, in 1953. He received the Electronic Engineering Diploma from PUC-RJ (Catholic University of Rio de Janeiro) in 1975 and his MSc and DSc degrees in Electrical Engineering from COPPE/UFRJ (Federal University of Rio de Janeiro) in 1982 and 1992, respectively. Since 1978 he has been working for the Federal University of Rio Grande do Norte – UFRN, at the Department of Electrical Engineering until 1999 and, lately, at the Department of Computer Engineering and Automation. He is a member of the Graduate Program in Electrical Engineering, where he is in charge of disciplines related to Computer Networks and orient the work of many graduate students. Since 1996 he has also been working as the Administrative and Planning Coordinator of the Point of Presence in Rio Grande do Norte (PoP-RN) of the Brazilian Research Network (RNP). His principal areas of interest are communication protocols, computer networks, mobile networks, distance education, formal description techniques, network engineering and Internet.