Performance analysis of long-lived cooperative transactions in active DBMS

Prithwish Kangsabanik *, D.S. Yadav, R. Mall, A.K. Majumdar

Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India

Received 22 July 2006; accepted 10 November 2006
Available online 14 December 2006

Abstract

Active database management systems (ADBMS) are used in different application domains and especially for cooperative and long duration activity management. This paper deals with performance analysis of long-lived cooperative transaction processing in an ADBMS. We first briefly discuss NP-QuadLock – a concurrency control scheme for cooperative and long durational transactions in ADBMS. A restricted version of NP-QuadLock named 2L-QuadLock has been used for simulation. We have modeled such an ADBMS supporting 2L-QuadLock scheme by a queuing model. The failure of the transactions running in such systems has been modeled by a failure recovery model. We have simulated this model for a transaction processing system serving long-lived and cooperative transactions. We also discuss some important emerging application scenarios, where the proposed cooperative complex transaction mechanism can be used (e.g. 3G-service environment, ubiquitous computing environment, feature composition in intelligent network environment, multi-site and multi-domain web-services).

An important objective of our work is to analyze quantitatively (a) the performance penalty on the system due to the partial abort, the number of locks held by a transaction, the number of states of the transactions, and (b) the gain in the performance of the system with the cooperation semantics proposed in 2L-QuadLock concurrency control mechanism. We have analyzed the effect of various parameters such as partial abort rate, cooperation rate, number of locks held by a transaction, multiprogramming level, on the performance metrics such as average service time, average saga length and the degree of compensation. Later, we characterize the application scenarios based on some important simulation parameters, and discuss the application performance needs for each of the application scenarios. The required performance parameters that need to be used for these application scenarios and the corresponding performance results using 2L-QuadLock are also discussed.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Active database; Concurrency control; Object oriented systems; Performance analysis; Queuing model

* Corresponding author. Present address: UBS AG, Postfach 8098, Zurich, Switzerland. Tel.: +41 79 653 1965.
E-mail addresses: prithwishk@yahoo.com, prithwish.kangsabanik@ubs.com, prithwish.kangsabanik@ieee.org (P. Kangsabanik), rajib@cse.iitkgp.ernet.in (R. Mall), akmj@cse.iitkgp.ernet.in (A.K. Majumdar).
1. Introduction

Traditional database systems are called passive because they execute transactions or query only when they are explicitly requested by the user or the application program. In contrast, active database systems are capable of executing transactions whenever some events occur. The occurrence of these events may take place due to changes in the database states. In active DBMS (ADBMS), transactions have the ability to monitor the database objects and to take specific actions when a particular event occurs in the database objects. This is specified by means of Event–Condition–Action rules or ECA rules. These events may be generated due to a transaction’s own object operation or due to the object operations generated by other transactions running in parallel. For some advanced database applications, it is necessary to monitor the conditions defined on the states of the database. As the conditions are satisfied, appropriate actions are invoked subject to the timing constraints. Examples of such applications, which are currently being developed using active databases, are hospital information system [1], Trading and stock control [2], Workflow management [3].

Increasing concurrency among transactions and maintaining the consistency of the database are two conflicting goals. This problem becomes more acute in ADBMS in the presence of long durational and cooperative transactions. The traditional notion of serializability as a correctness criterion turns out to be a bottleneck for long durational and cooperative transactions. In order to overcome this bottleneck, different kinds of transaction models e.g. Saga [4,5], nested transaction [6], cooperative transactions [7] have been developed. These transaction models generally use relaxed notion of correctness i.e., ACID (Atomicity, Consistency, Isolation and Durability) properties.

1.1. Long durational transactions and sagas

A transaction, which takes relatively long time to complete its execution even without any interference from other transactions, is known as Long Duration Transaction or Long-Lived transaction (LLT). Since an LLT takes a long time to finish its computation, it is more vulnerable to failures [8]. In addition, if an LLT retains all the locks acquired by it until it commits, a large amount of work has to be undone in the event of abort. This makes the recovery procedure more complicated and costly affair. An LLT imposes a relatively long waiting time for short transactions waiting for the locks acquired by the LLT. Thus, an LLT imposes serious constraints on the overall performance of the system.

A Saga [4,5] is a long durational transaction model that can be expressed as a series of subtransactions, called component transactions, which can be interleaved with other concurrently executing component transactions. If any activity needs to be aborted or partially aborted, a compensatory action may be taken for committed component transactions. These transactions are known as compensating transactions. Sagas provide high degree of concurrency by releasing the locks at the commit of the component transactions. A component transaction is executed atomically and it retains the locks until commit, following the two-phase locking protocol. Since Sagas are also LLTs, the overhead required to restore data consistency in case of failures cannot be avoided. However, Sagas support forward progress of activities.

Using an ADBMS, we can model long-lived transactions, where a transaction can be viewed as a collection of subtransactions. Execution dependencies among the subtransactions are not only defined in terms of significant events e.g. (Begin, Commit, Abort) of subtransactions but also in terms of monitored database object events.

An ADBMS transaction may interact with other concurrent transactions by making its changes in the database accessible to other concurrently running transactions. Thus, a transaction processing system requires a controlled cooperation among these concurrent transactions. This means that before the commit of a transaction, the effects of execution of the transaction on the database will be visible only to those transactions that can cooperate with it.

In our earlier work, the execution of long durational and cooperative transactions has been investigated in [9,10], where a concurrency control scheme named NP-QuadLock has been proposed. NP-QuadLock exploits the semantics of the transactions to achieve better cooperation and concurrency among the ADBMS transactions. It gives a suitable criterion for “correctness” of execution of ADBMS transactions in the presence of inter-transactional events and detached mode ECA rules.
In this paper, we build on that work by focusing on the performance analysis of NP-QuadLock scheme by means of simulation. The purpose of this simulation study is to analyze the performance of NP-QuadLock scheme quantitatively due to the partial aborts, number of locks held by a transaction, number of states in a transaction, and degree of multiprogramming. The NP-QuadLock locking scheme has the ability to handle nested and concurrently fired (sub)transactions. It also supports compensation of the (sub)transactions by means of compensating transactions. In our simulation studies, we have used a restricted version of NP-QuadLock scheme with two levels of nesting. We call this restricted NP-QuadLock scheme as 2L-QuadLock. The transactions generated by NP-QuadLock or 2L-QuadLock schemes are called NP-CT or 2L-CT transactions respectively. We have shown that the proposed 2L-QuadLock scheme behaves like a Saga [4,5,11,12] when the cooperation among the complex transaction types is very high. On the other hand, with low cooperation rate the transactions in the present scheme behave like classical atomic transactions. Thus, 2L-QuadLock scheme can be adopted to deal with atomic transactions as well as long-lived transactions by varying the degree of cooperation among the complex transactions.

Thus the goals of our simulation study are:

- Evaluate quantitatively the proposed 2L-QuadLock scheme with different cooperation rates, transaction sizes, conflict rates, fault rates and multiprogramming level.
- Study the efficacy of this scheme in presence of long durational and cooperative transactions in active DBMS.
- Show the effectiveness of the 2L-QuadLock scheme in other application scenarios, for example, in 3G service environment, ubiquitous computing environment, Multi-site workflows, Multi-site web-services, batch applications and batch utilities, and feature composition in Intelligent Network environments.

We have developed an event-driven simulator to study the performance of 2L-QuadLock under different fault conditions and cooperation rates among the complex transactions.

Our simulation study is based on a queuing network model which captures the primary features of long durational and cooperative transaction processing systems namely data locking, transaction cooperation, resource contention, and failure recovery. The simulation experiments have been carried out by varying several parameters such as number of locks per transaction, cooperation rate, failure rate, and multiprogramming level. The simulation experiments indicate that complex transactions with high cooperation rates outperform complex transactions with low cooperation rates over a wide range of multiprogramming level, failure rates, and number of locks to be acquired by the transactions. This reinforces our confidence on the proposed locking schemes for long duration and cooperative transactions.

1.2. Related work

The performance analysis of the locking protocols had received considerable attention in past years. However, most of these investigations [13–18] concentrated on one type of concurrency control policy such as locking, time-stamp, etc., and assumed there are no precedence constraints among transactions. Menasce and Nakanishi [19] and Morris and Wong [20] have studied the trade-off between the optimistic and pessimistic concurrency control methods, but their models are applicable only to a system with a single class of transactions, with no consideration of precedence constraints and cooperation among the transactions. The performance analysis of long-lived transaction processing system in presence of rollbacks and aborts has been investigated by Liang and Tripathi [8].

Active database systems with ECA rules have been found to provide an elegant framework for capturing semantics of many real-life applications which are long durational and cooperative in nature [21,22]. Extended transaction models have been proposed to define the semantics of rule evaluation [23]. In [23,1,24], an extended nested transaction model has been proposed to define the semantics of rule evaluation of HIPAC – an Active Relational database. In other active relational databases like Starburst [25,26] and Heraclitus [27], the rules are fired as part of the event generating transaction, with different execution semantics defining order of execution of rules and the methods to compose the updates of the rules. ODE, an active object oriented DBMS (AODBMS) has used an extended nested transaction model proposed in [28] for executing rules.
In all of these schemes, the action part of the rule is always fired as part of the event generating transaction, i.e., the events are always intra-transactional. In another AODBMS named Sentinel [29,30], an effort has been made to take care of inter-transactional events, i.e., where transaction generated events cross transaction and application boundaries. Thus, a transaction can take action using an event generated by another concurrently running transaction belonging to a different application.

In some AODBMS, the long duration and cooperative transactions are modeled as a set of ECA rules with different modes of coupling [23]. The flexibility of rule execution is guaranteed through the definition of various coupling modes that define the execution of rules or parts thereof relative to the triggering transactions. HIPAC recognized four coupling modes, namely immediate, deferred, detached and parallel detached causally dependent. In [31,32], it was shown that for closed database applications these coupling modes are sufficient.

For open environments, i.e., situations in which rules may cause non-recoverable side effects and also provide cooperation among the transactions, two additional coupling modes are required, namely sequentially detached causally dependent and exclusive detached causally dependent [31,32]. However, the execution semantics of AODBMS transactions for this kind of open environments still do not exist. For this kind of open environment, the open nested transactions as suggested by ODMG [33] can provide suitable execution framework for AODBMS transactions. Thus, using ECA rules one can define the control flow and cooperation semantics within the subtransactions of an open nested transaction. The application systems currently being developed with active databases like hospital information system [1,24], trading and stock control [2], workflow management [3], etc. require transaction models which are of long-duration and cooperative in nature. These applications also require monitoring capability of the database — to take particular action (or transaction firing) depending on the current state of the database. A running transaction may interact with other concurrently running transactions by making its changes in the database accessible to other transactions running in parallel. Thus, these transactions require a controlled-cooperation among them.

The main problem with ECA rules is that they do not have a concept of correctness when the inter-transactional events and detached mode rules are taken into account. Thus, it is impossible to determine if an execution of a set of transactions is “correct” in presence of detached mode ECA rules and inter-transactional events. Also using ECA rules, we cannot define how transaction of one type may cooperate with other transaction types. In this paper, we address these issues and define a concurrency control mechanism for such type of long duration and cooperative AODBMS transactions.

The preceding discussions reveal that none of the AODBMS mentioned above support horizontal cooperation, i.e., where a transaction’s effect may be read by other transactions running in parallel, based on some user defined transaction cooperation semantics. Vertical cooperation, i.e., parent–child cooperation is supported by some execution models based on extension of nested transaction model. It was pointed out in [34,35] that using horizontal cooperation, cooperative applications like workflows, cooperative editing, etc. can be handled. It was argued in [36], that how far ECA rules should be allowed to influence the application program need to be carefully analyzed. On the one hand, one can argue that database systems should not be seen as a mere slave to the application, but rather should be able to autonomously control the application activities in various ways [34,35]. On the other hand, it should also be possible that the application need not be aware of the underlying active capabilities if it chooses to do so.

In our work on NP-QuadLock-based concurrency control scheme, we have delved into these aspects. In this paper, we do a performance analysis of the concurrency control scheme 2L-QuadLock, a restricted version of NP-QuadLock, for cooperative and long duration transaction management in AODBMS based on an open nested transaction framework. NP-QuadLock exploits the semantics of the transactions to achieve better cooperation and concurrency among the transactions. The NP-QuadLock scheme helps in defining the execution semantics of AODBMS transactions in presence of inter-transactional events and detached mode ECA rules.

1.3. Organization of the paper

This paper is organized as follows. We discuss some preliminary concepts concerning complex transactions and base transactions and provide the necessary definitions in Section 2. The compensation issues of the 2L-CT complex transactions are discussed in Section 3. We discuss some example application scenarios, where
2L-CT complex transactions can be used in Section 4. The locking mechanism for 2L-CT transactions are discussed in Section 5. In Section 6, we discuss the queuing model of the database system which we have used in our simulation of 2L-QuadLock scheme. The important parameters that we have used in the simulation are discussed in Section 7. The fault model of a complex transaction type has been discussed in Section 8. Section 9 presents the simulation results obtained in different fault conditions, multiprogramming levels, number of locks acquired by the transactions, and cooperation rates. In Section 10, we characterize different application scenarios based on some important simulation parameters, and, discuss the application performance needs for each of these application scenarios. We show how the simulation parameters set-up (presented in Section 9) relates to application scenarios (of Section 4). The performance results achieved with the corresponding simulation parameters, for each of the application scenarios, are also discussed in this section. Section 11 concludes this paper and provides direction for future works.

2. Preliminary concepts

In this section, we discuss some concepts crucial to our work and introduce some definitions and notations. An ECA rule has the following structure:

\[ \text{rule} \{ \text{rulename} \} : \]
\[ \quad \text{on} \{ \text{event expression} \} \]
\[ \quad \text{if} \{ \text{condition expression} \} \]
\[ \quad \text{then} \{ \text{action expression} \} /* \text{e.g., a base transaction firing} */ \]
\[ \quad \text{rulemode} \]

where

- Event expression: The event expression is defined in terms of observed database object events generated by the transactions.
- Condition expression: It denotes the precondition for the execution of the base transaction. It is defined in terms of the database object queries.
- Action expression: This defines the effect of firing a base or complex transaction. In the action expression, a set of object operations can be executed as a part of a base transaction. A set of object operations, which compensates the effects of a base transaction, is also defined. This set of object operations is called a compensating base transaction (CBT) for the corresponding BT. We may express the firing of a complex subtransaction in the action expression. Similarly, we may express firing of a set of base and complex transactions in parallel in the action expression. Thus, an action expression, \( \langle \text{ActExpr} \rangle \) written in BNF has the following form:

\[ \langle \text{ActExpr} \rangle : \{ \langle \text{BT-Expr} \rangle \mid \{ \langle \text{CT-Expr} \rangle \} \} \\text{ParBegin} \{ \{ \langle \text{BT-Expr} \rangle \mid \{ \langle \text{CT-Expr} \rangle \} \} \} + \text{ParEnd} \]
\[ \{ \langle \text{BT-Expr} \rangle : \{ \langle \text{BT-definition} \rangle \} \mid \{ \langle \text{CBT-definition} \rangle \} \} \mid \text{fire \langle BT-Type-id \rangle} \]
\[ \{ \langle \text{CT-Expr} \rangle : \text{fire \langle CT-Type-id \rangle} \} \]

With a complex transaction type, a compensating base transaction is also defined. The CBT contains a set of object operations that needs to be executed when this CT needs to be compensated.

- Rulemode: Rulemode specifies the type of the ECA rule and is of the following types: (i) immediate, (ii) deferred and (iii) detached. The rulemode specifies the coupling between event–condition and action part of the rule. The rules with immediate and deferred rulemodes are also called intra-transactional rules, since the event expression of an intra-transactional rule is evaluated with the events generated within a single transaction. The rules with rulemode detached are treated as inter-transactional rules. The event expression of a detached mode rule is evaluated with the events coming from different committed atomic transactions. When a complex transaction is fired by a rule or a set of base/complex transaction is fired in parallel then the rulemode is always detached. An additional rulemode (known as coupling mode) sometime also used to distinguish the coupling between the event and condition. This additional event–condition coupling mode can also be of the types: immediate, deferred, and detached. The immediate and deferred coupling modes of event and action are used in intra-transactional rules. The detached coupling mode of event and condition is used in inter-transactional rules. Here, condition and action of a detached mode rule with detached mode
coupling is normally executed in a single atomic transaction. The transaction designer uses the detached mode rules to define the state transition graph (STG) of the complex transaction type, which will be discussed below.

In the action part of the rule, a previously defined base/complex transaction can be fired. In addition, in the action part, a new base transaction schema may be defined and this new transaction may be fired. Therefore, the action part of the rule of a complex transaction may either fire an instance of a predefined complex transaction type or may define a new base transaction and fire it. In the action part, a number of base and complex transactions may also be fired in parallel.

The base transactions are used to define the atomic ADBMS transactions. Complex transactions are used to define the long durational and cooperative ADBMS transactions. Thus, the complex transaction types help in defining the open nested transactions in an ADBMS.

2.1. Base transaction

A base transaction (BT) defines a collection of database object operations, which have to be executed as an atomic transaction. A base transaction may trigger immediate and deferred mode rules. These triggered rules are executed as a subtransaction of a base transaction. The entire base transaction is executed following a nested transaction model [6]. The root of the base transaction commits atomically to the database and the triggered subtransactions are committed to its parent (sub)transaction. The condition–action part of a detached mode ECA rule is also treated as a base transaction, when it does not fire a complex transaction in the condition–action part.

2.2. Complex transaction

A complex transaction is associated with a complex transaction type, which defines the schema of the complex transaction. The complex transaction types are formed from a collection of base and complex transactions, a set of detached mode ECA rules and a state transition graph (STG). The condition–action part of an ECA rule defines the subtransactions of a complex transaction. A subtransaction may be a base transaction or another complex transaction (CT). The STG of a complex transaction type defines a set of states through which the complex transaction may go through during its lifetime, with a designated start state, a set of final states, and a set of state transitions among these states. A state transition is labeled by a rule from the ECA rule set. Each complex transaction is associated with a state during its execution. A complex transaction moves from one state to another due to the occurrence of the events specified in the rule and the successful or failed execution of the corresponding ECA rules. We associate two separate destination states corresponding to a successful and a failed execution. A complex transaction finishes its execution when a final state is reached. Within an ECA rule, a number of base/complex transactions may be fired in parallel. Thus, the STG of a complex transaction type specifies the set of correct execution sequences of the base/complex transactions. The STG of a complex transaction type facilitates in creating event-driven transactions. An additional cooperation parameter is associated with each state in the STG for specifying different complex transaction types that can interleave between the executions of two base/complex subtransactions of the given complex transaction.

We associate a compensating base transaction (CBT) with each complex transaction type, which compensates the effect of a complex transaction, in case the parent of the complex transaction decides to compensate its computation partially or fully. The role of a compensating transaction is similar to that in Saga [4,5,11]. We know that a Saga is a set of relatively independent (component) transactions $T_1, T_2, \ldots, T_n$, which can interleave with component transactions of other Sagas. Component transactions within a Saga execute in a predefined order, which can be either sequential or parallel. In order to handle transaction abortion, a component transaction $T_i$ may be associated with a compensating transaction $CT_i$. A compensating transaction $CT_i$ undoes, from a semantic point of view, any effect of $T_i$, but does not necessarily restore the database to the same state that existed when $T_i$ began executing. Both component and compensating transactions work like atomic transactions and enforce the changes on the stored objects at the commitment time. Thus, isolation is limited to the component transaction level and the Sagas may view the partial results of other Sagas. A Saga
commits, i.e., successfully terminates, if all of its component transactions commit in the prescribed order. Under sequential execution, the correct execution of a committed Saga is: $T_1, T_2, \ldots, T_n$, i.e., all component transactions complete sequentially in proper order. A Saga may also abort partially/fully as defined by the Saga designer, in case a component transaction of the Saga aborts. A partial abort is done by running compensating transactions for some of the committed subtransactions and after that Saga can again proceed with its computation. In case of a full abort, the entire Saga is compensated by running compensation subtransactions for each of the committed component transactions.

As in Saga, in our model a complex (sub)transaction may decide to abort partially/fully by running the CBTs associated with the committed subtransactions. Thus the sequence of base/complex subtransactions that have been fired due to state transitions from the start state to the current state have to be compensated. We restrict the number of subtransactions that need to be compensated by designating some of the states in the STG as marked states. The marked states signify the checkpoint marks of the complex transaction computation. In case of a partial/full abort of a complex (sub)transaction, compensation is done for only those committed subtransactions, which are fired after the Last Visited Marked State (LVMS). Thus, the sequence of transitions that have been fired after the LVMS, are compensated in reverse chronological order. After compensation, the complex transaction can proceed with its computation from the state reached.

The subtransactions are executed by the parent complex transaction according to its STG. The execution dependencies among the subtransactions are defined in terms of significant events happening within the subtransactions (e.g. Begin, Commit, Abort, PrepareToCommit, etc.) and the monitored database object events. Here monitored database object events may be local or global, intra-transactional or inter-transactional.

A complex transaction (CT) is a fired instance of a complex transaction type. We denote the transaction type of a complex transaction $T_k$ by $ty(T_k)$. In the following section, we give the formal state transition model for defining such complex transaction types.

2.3. State transition graph (STG) of a complex transaction type

The STG of a complex transaction type $t$ is specified using the formal state transition model of a complex transaction type as given in [10,9]. It has a set of states $S$ with a designated start state ($s_0$), a set of successful and failed final states, a set of detached mode ECA rules $RuleSet$, and the state transition functions $\delta$ and $\bar{\delta}$ over $S$ corresponding to “successful” and “failed” execution of the rules from the $RuleSet$. The set of nodes of this graph are the set of states $s \in S$. The set of edges are the transitions in $\delta$ and $\bar{\delta}$ of the form $(s_i, r_k, s_j)$. A transition is represented by a directed edge from the node $s_i$ to the node $s_j$ in STG and the edge is annotated with a rule $r_k \in RuleSet$. A failed transition $(s_i, r_k, s_j) \in \bar{\Delta}$ iff $\exists (s_i, \bar{r_k}, s_j) \in \Delta$. A transition $(s_i, r_k, s_j) \in \Delta$ is said to be vital iff the corresponding failed transition $(s_i, \bar{r_k}, s_j)$’s final state $s_j$ is a “failed” final state of $S$. The STG of a complex transaction type defines the following:

- How the different instances of this complex transaction type change states because of successful and failed execution of the detached mode ECA rules.
- The cooperation semantics of this complex transaction type with other complex transaction types.

A complex transaction $T$ may allow showing its effect on database to some other cooperating complex transactions before it commits. This cooperation semantics is specified by associating a set of complex transaction types with each state of $T$. In a state $s$, the effect of a complex transaction $T$ becomes visible to other complex transactions whose types are specified in the state $s$.

Let’s us assume that the complex transaction (say $T$) is in state $s_1$ currently and does not modify again the same data (say $d$), which it previously accessed. Then in the new state of $T$ (say $s_2$), other transaction types which were allowed to see data of $T$ in state $s_1$ will still be able to see $d$ in addition to those transaction types which are permitted in $s_2$. If $T$ modifies $d$ in the transition $(s_1 \rightarrow s_2)$ then only the transaction types allowed seeing data at $s_2$ will be able to see $d$. Thus another complex transaction $T'$ can see the effect of $T$ when $T$ is in state $s_2$ provided $T'$ has a type specified in the cooperation parameter of $s_2$.

A complex transaction can therefore be viewed as a collection of related base and complex transactions, which are activated because of the firing of the detached mode ECA rules. A complex transaction during
its execution may fire in parallel base/complex subtransactions. The execution of a complex transaction generates a **Transaction Tree** in which the intermediate nodes are different complex subtransactions and the leaf nodes are different BT or CBT. The execution of such leaf level BT/CBT generates new events, which can be subscribed to by other complex transactions.

In order to illustrate the above concepts regarding complex transaction, let us consider a simple example. Let there be three complex transaction types $\text{CTtype}_1$, $\text{CTtype}_2$ and $\text{CTtype}_3$. The STGs of $\text{CTtype}_1$, $\text{CTtype}_2$ and $\text{CTtype}_3$ are shown in Fig. 1. The different edges of the STGs are labeled by their corresponding rules and the subtransactions that are fired by those rules. The dotted edges specify the state transitions in $\delta$, i.e., the state reached when the corresponding transition in $\delta$ fails. For example, the transition $(s_{11}, R_{12}, s_{12})$ of $\text{CTtype}_1$ fires two complex transaction $\text{CT}_2$ and $\text{CT}_3$ in parallel, which are instances of complex transaction types $\text{CTtype}_2$ and $\text{CTtype}_3$ respectively. Also this transition is vital since in the transition $(s_{11}, \overline{R}_{12}, s_{15})$, $s_{15}$ is a failed final state.

Now in order to illustrate the transaction tree, let us assume a complex transaction $\text{CT}_1$, which is an instance of $\text{CTtype}_1$, has been fired. $\text{CT}_1$ is currently in state $s_{11}$ and rule $R_{12}$ is currently fired by $\text{CT}_1$. Thus $\text{CT}_2$ and $\text{CT}_3$ execute in parallel as subtransactions of $\text{CT}_1$. Let $\text{CT}_2$ and $\text{CT}_3$ be in states $s_{20}$ and $s_{31}$ respectively. $\text{CT}_2$ has fired the rule $R_{21}$ causing the firing of $\text{BT}_3$ and $\text{BT}_4$ in parallel. $\text{CT}_3$ has fired the rule $R_{32}$ causing the firing of $\text{BT}_7$ and $\text{BT}_8$ in parallel. The current transaction tree of $\text{CT}_1$ is shown in Fig. 2. Here we see that all the intermediate nodes in the transaction tree are complex transactions and the leaves are base transactions.

### 2.4. NP-CT complex transaction

A complex transaction may fire complex and base subtransactions in parallel. Thus, it forms a transaction tree with intermediate nodes as complex subtransaction and the leaf nodes as base transactions. These types of complex transactions are called as **NP-CT complex transactions**.

![Fig. 1. State transition diagram of different complex transaction types.](image-url)
2.5. 2L-CT complex transaction

A complex transaction is said to be a 2L-CT complex transaction when it fires only base transactions as subtransactions and no subtransactions are fired in parallel. Thus, in this case, the complex transaction only fires the base transactions in sequence. Here, the complex transaction and its base subtransactions form a two level transaction tree where at the first level (root level) there are complex transactions and where at the second level (leaf level) there are base transactions.

3. Compensation of 2L-CT complex transactions

The compensation model for NP-CT complex transactions and a correctness criterion for it have been proposed in [9]. This correctness criterion also takes care of cooperation among the complex transactions. The main advantage of this approach is that it allows reasoning about histories with compensation operations, using only knowledge about the cooperation and compensation semantics as specified by the user, without reference to the semantics of database states. Since 2L-CT transactions are a restricted version of NP-CT complex transaction, thus compensation procedure for 2L-CT complex transaction is same as that of the NP-CT complex transaction but with less complexity. In the following, we discuss briefly about the compensation of 2L-CT complex transaction.

3.1. Compensation of a committed state transition in 2L-CT

When a state transition $tt$ of a 2L-CT complex transaction has been committed, later $tt$ can be compensated by running the CBT associated with $tt$, i.e., the CBT associated with the fired BT in $tt$.

3.2. Full abort of 2L-CT complex transaction

A 2L-CT complex transaction (T) may need to be “fully” aborted during its execution because of the following reasons:

- A fired vital transition (say $t$) of $T$ is aborted due to the abort of the corresponding BT.
- The complex transaction itself aborts.

A complex transaction is said to have “failed” when a “full abort” occurs. In case of a full abort of a 2L-CT complex transaction, the following actions are taken:

![Complex transaction tree](image-url)
The active subtransaction corresponding to the currently fired rule is aborted.

Next the sequence of committed state transitions, which have been fired after the last visited marked state (LVMS), are compensated by executing the CBTs associated with each of the transitions. The sets of CBTs are invoked in reverse chronological order of the execution of the transitions, which have been fired after the LVMS.

At this stage, the complex transaction makes transition to a failed final state.

3.3. Partial abort of 2L-CT complex transaction

A 2L-CT complex transaction may have to be partially’’ aborted when one of its fired nonvital transitions is aborted. The CBTs to be executed in case of partial abort are similar to that in the case of full abort. However after partial abort, a 2L-CT complex transaction can proceed with its computation from the state \( s \) it had reached after partial abort.

4. Example application scenarios of 2L-CT complex transaction

In this section, we will show some example application scenarios, where our 2L-CT complex transaction scheme in ADBMS will be of use. The chosen application scenarios are the following:

- Composition of cooperative mobile transaction in 3G service environment.
- Composition of multi-site web-services across multiple resource domains.
- Multi-site workflows and Multidatabase transactions using cooperating ADBMS transactions.
- Controlled resource cooperation among concurrently running batch applications.
- Feature composition in Intelligent Networks (IN) applications.
- Control flow cooperation in Ubiquitous computing applications.

4.1. Composition of cooperative mobile transaction in 3G service environment

The evolution of third-generation (3G) mobile communication networks has enabled the need for access of shared data/services to provide new service offerings to mobile users. Using new wireless mobile devices, it is now possible to provide users with the ability to retrieve/update information that is stored at different sites/databases. Mobile transactions executed by the mobile users are inherently long durational in nature due to disconnected network activity and limited battery life of mobile devices. These mobile transactions access composed transactional services, where each transactional service may access one or more local databases.

Here we propose a mobile transaction management scenario using 2L-CT complex transaction scheme, which can ensure:

- the mobility aspect of the transactions moving from one Radio Network Controller (RNC) to another (driven by the user movement from one Cell to another),
- handling of the long durational nature of the mobile transactions, where the longevity may be due to the disconnection as well as due to the application semantics, and,
- cooperation among the mobile transactions, where individual transactions may share data objects with other concurrent transactions during their execution.

In the following subsections, (a) we present the mobile computing environment in 3G environment and (b) we present usage of our proposed 2L-CT complex transaction model in mobile transaction environment supporting long durational, nested, and cooperative mobile transactions accessing different site databases in a transactional manner.
4.1.1. The 3G mobile computing environment

The mobile computing environment for the third generation mobile network is shown in Fig. 3. In this environment, a mobile transaction is fired from the Mobile Hosts (MH) to the Mobile Switching Center (MSC), which forwards the request to the Mobile Transaction Manager (MTM) residing at the IP network. MTM has the schema of the corresponding mobile transactions and executes the transactional Value Added Services (VAS) at each individual site based on this schema.

In 3G mobile networks, entire geographical coverage of a service provider is divided into number of cells. A service provider’s network is composed of a Core Network and a number of Radio Access Networks (RAN). Each RAN takes care of a number of cells through (a) a Radio Network Controller (RNC), and, (b) a number of Base Stations (BS) controlling the mobile hosts at each cell. In the Core Network, there are Mobile Switching Centers (MSC), which connect to the RNCs and handles the handoff of the mobile hosts, as it moves from one cell to another, thereby providing a continuous connection. The Global Mobile Switching Center (GMSC) supports the interoperation with other service providers in Public Land Mobile Network (PLMN), Public Switched Telephone Networks (PSTN), and as well as Internet. The Valued Added Service Providers (VASP) provide the VAS-based services, and it is added to the Service Control Point (SCP) when the service is introduced. The composition of VAS-based services giving proper transactional and cooperation semantics are provided by the MTM. The MTM itself can be thought of as a VAS and will be registered in SCP. Individual Mobile subscribers can provision the VASs or composed VASs.

4.1.2. Usage of 2L-CT complex transaction model in 3G environment

We address the issue of composition of VAS-based services and propose a transaction model for nested and cooperative transaction management in mobile environment using 2L-CT complex transaction. With this objective, we introduce the concept of Complex Transaction Types (CTT) by means of a set of ECA (Event Condition Action) rules and a State Transition Model (STM). The condition–action part of the ECA rules
constitutes the subtransactions of the complex transaction. A subtransaction may be an atomic transaction, called Base Transactions (BT) or another Complex Transaction (CT). The value added service provider provides the BTs, which execute transactions at autonomous Local DBMSs (LDBMS). The service interface may be based on Web-Services, if the services are deployed over Internet.

Within an ECA rule of a CTT, a number of base/complex transactions may be fired in parallel. The state transition model of a complex transaction type specifies the set of correct execution sequences of the base/complex transactions. An additional cooperation parameter is associated with each state in the State Transition Graph (STG) of a STM, for specifying complex transaction types that can interleave between the executions of two base/complex subtransactions of the given complex transaction. Also we associate compensating base transactions (CBT) with each complex transaction type and base transactions, which compensates the effect of the complex transaction and base transaction in the VAS sites, in case the parent of the complex transaction decides to compensate its computation partially/fully.

The execution of a complex transaction generates a Transaction Tree in which the intermediate nodes are different complex subtransactions and the leaf nodes are different BTs or CBTs. The concept of state associated with a complex transaction is also of use here. The state of a complex transaction identifies which events a complex transaction can subscribe to and which other complex transaction type can cooperate with CT. Because of firing of ECA rules, complex transaction undergoes state transitions.

Thus, we see that our proposed 2L-CT complex transaction is also of use in mobile computing environment.

4.2. Composition of multi-site web-services across multiple resource domains

In Web-services based Service Oriented Architecture (SOA), a business process can be described as a composition of several base web-services. The individual base web-services are atomic and the composite web-service representing the business process may execute across several resource domains within and across organizational boundaries [37–40]. The requests from end-users to execute different composite web-services may need to be executed in parallel and therefore need to share the resources (e.g. DBMS being one of the resources in addition to CPU, memory, files, etc.) during the life-time of their execution. What will be the correctness criterion when such composed web-services are scheduled over multiple resource domains? Therefore, we need a way to specify the correctness criterion of the composite web-services. The proposed complex transaction modeling approach can provide a cooperation specification to the composite web-service scheduler, so that resource sharing policies across multiple resource domains can be specified using the proposed transaction modeling framework.

4.3. Multi-site workflows and multidatabase transactions using cooperating ADBMS transactions

In a multi-site workflow or a multidatabase complex transaction scenario, the correctness of concurrently running workflows can be specified using the proposed cooperating transaction specification framework. The multi-site workflow manager or multidatabase transaction manager can work as a site scheduler controlling firing of individual workflow steps or site transactions at the multidatabase sites depending on the specified cooperation semantics of different workflow types or multidatabase transaction types.

4.4. Controlled resource cooperation among concurrently running batch applications

In a batch-processing environment, different resources (e.g. files, databases, MQs) need to be locked for long duration within the different batch steps and across the batch process. Using the proposed transaction cooperation mechanism, these resources can be released to other batch job types in a controlled manner. For example, as shown in Fig. 4, after a batch step in a particular batch job type, different other batch job types who can share the files being accessed by the preceding batch steps can be specified.

In a modified scenario (see Fig. 5), if the above batch jobs are accessing shared Message Queues (MQs), then the proposed cooperation specification mechanism will enable us to specify controlled publish-subscribe among the batch job types. Here, we will get the additional benefit of a publisher controlled subscription
mechanism, i.e., which other batch job types can consume the produced message can be specified by the publisher itself.

4.5. Feature composition in Intelligent Networks (IN) applications

In IN applications in telecommunication environment, the new features that will be added in future are not known at design and development time. This results in a feature interaction problem. New feature types meet each other at run-time. Therefore, if we want to guarantee a controlled feature interaction and a consistent composite feature, the cooperation semantics for individual features still need to be specified for controlled cooperative interaction among them. With the proposed transaction cooperation specification mechanism, the features can be modeled as complex transaction types and the allowable cooperation semantics of individual features can be specified before hand, enabling a controlled interaction among the existing and future features.

4.6. Control flow cooperation in Ubiquitous computing applications

In a ubiquitous computing environment, the applications running within a Ubiquitous Computing Entity (UCE) sometimes need to cooperate their control flow with applications running in other ubiquitous
computing entities (e.g. in PatchPanel [41]). Here also, the UCEs and the applications running inside them meet only at the run-time. In this scenario, each individual UCE application may publish their cooperation semantics with other UCE applications to a central scheduler. Depending on the cooperation semantics of the ubiquitous computing entities, the central scheduler may take care of the publication of events coming from other ubiquitous computing entities. Thereby, control flow cooperation among the ubiquitous computing applications can be made possible using our proposed cooperative complex transactions.

5. 2L-QuadLock: a concurrency control scheme for 2L-CT complex transaction

The 2L-QuadLock mechanism uses a modified version of Moss's locking algorithm for nested transactions [6]. The locking algorithm (i) ensures serializable schedule of individual BTs, (ii) produces a correct history according to cooperation semantics of the CT types, and (iii) handles abort and partial abort of the complex transactions.

The individual BTs acquire locks with their own transaction-ids. A BT acquires locks in a two-phase manner by requesting the global scheduler. A BT can potentially access any object that has previously been accessed by one of the base transactions of its parent. This is achieved since a BT is allowed to access an object whose lock is currently held by the parent CT. At the commit of a BT, the effects of the corresponding transactions are reflected on the database and other BTs can see the modified objects according to the cooperation semantics of the parent CT. During commit of a BT, locks acquired by it are inherited by the parent CT. In addition, at the same time, permission is given by the parent CT to share the locks with other cooperating complex transactions according to the cooperation semantics of the state being reached within the parent CT. A CT is said to be committed when it reaches one of the successful final state.

The partial/full abort of a complex transaction $CT_i$ may be due to the abort of a fired BT of it. To abort a complex transaction partially/fully, all the CBTs corresponding to the transitions from the LVMS of $CT_i$ has to be executed in reverse chronological order. CBTs acquire locks as a normal BT. While releasing the locks acquired by a CBT, all the locks acquired by the CBT are released instead of being inherited by the parent $CT_i$. After all these CBTs are committed, the locks that are inherited by $CT_i$ after the LVMSs are released. The CT proceeds with its computation from the state being reached in case of the partial abort. In case of complete abort of a CT, all the locks that have been acquired by the CT are released.

5.1. Locking mechanism in 2L-QuadLock

The proposed locking mechanism uses four modes of locks [42], denoted as shared (S), exclusive (E), relatively shared (RS) and relatively exclusive (RE). Locks of type S and E have meaning similar to those used in two-phase locking scheme [43]; that is these locks are used to ensure that individual BTs are executed atomically. Locks of type RS and RE are used to produce non-serializable, cooperative interleaving among the CTS.

The relationships among four modes of locks can be summarized as follows. Each BT $BT_{ij}$ of a CT $CT_i$ must obtain a S-lock or a E-lock on a database object $x$ before reading or writing $x$, respectively. The locks acquired by a BT during its execution is inherited by the parent CT (after changing the mode) during the commit of the BT. If a BT $BT_{ij}$ of $CT_i$ acquires an S-lock on $x$ during its execution, and it does not update $x$ within $BT_{ij}$, then this S-lock is modified to an RS-lock when $BT_{ij}$ commits and it is inherited by the parent $CT_i$. Otherwise, if $BT_{ij}$ updates $x$ after reading it inside the same BT (or in a BT $BT_{ik}$ of $CT_i$ invoked earlier) then the S-lock (the RS-lock) is modified to an E-lock at the time of write operation of $BT_{ij}$ on $x$. Similarly, after obtaining an E-lock on $x$ during the execution of $BT_{ij}$ this lock is also modified to an RE-lock when $BT_{ij}$ commits and it is inherited by the parent complex subtransaction $CT_i$. The reason behind modifying the modes of lock and getting it inherited by the parent CT is to allow nonserializable cooperative interleaving among the complex transactions.

A complex transaction $CT_i$ allows the RS/RE locks to be shared by other CTs according to the state being reached. This is done by maintaining a set called interleaving set of $CT_i$ on $x$, which is denoted as InterleaveSet ($CT_i$, $x$). This set contains those CT types whose instances are allowed to interleave with $CT_i$ on $x$ i.e., that can access $x$ concurrently with $CT_i$. This information is used by the scheduler to decide on the compatibility of various modes of locks.
5.2. Compatibility among locks in 2L-QuadLock

The compatibility among various modes of locks is shown in Fig. 6. The set $\text{InterleaveSet}(CT_i, x)$ is initialized as empty when a base transaction of $CT_i$ obtains an S-lock or an E-lock on $x$. More complex transaction types are added as each base transaction of $CT_i$ commits and thus passes through different states in the STG of $CT_i$. In the compatibility matrix shown in Fig. 6, “Y” means that the lock requested by a base transaction $BT_i$ is compatible with the lock held by a base transaction $BT_j$ or a complex transaction $CT_k$; that is lock request can be granted. Similarly, the symbol “N” means that the lock requested by $BT_i$ is incompatible with that held by $BT_j$ or $CT_k$ and therefore cannot be granted. The symbol “C” means that the lock requested by $BT_i$ may or may not be compatible, depending on whether the $\text{ty}(CT_i)$ is contained in the interleaving set of $CT_k$ on $x$. If $\text{ty}(CT_i) \in \text{InterleaveSet}(CT_k, x)$, then the lock requested by $BT_i$ is compatible i.e., $C = Y$; otherwise lock requested by $BT_i$ is incompatible i.e., $C = N$.

This locking algorithm in 2L-QuadLock [10,9] ensures the serializable schedule of individual base transactions, produces correct history according to the cooperation semantics of complex transaction types and handles aborts and partial aborts of transactions.

In the following section, we discuss the queuing model of the ADBMS system supporting the 2L-QuadLock scheme.

6. The queuing model

In this section, we present a queuing model of an ADBMS, which follows the 2L-QuadLock scheme. The ADBMS is modeled by a network of queues and services as shown in Fig. 7. It consists of a set of terminals, a transaction manager (TM), a lock manager (LM), a TM waiting queue (TMQ), a LM waiting queue (LMQ), a set of query servers, and a fault diagnosis center. The closed queuing system comprising of LM, LMQ, query servers, and fault diagnosis center, is called as a Transaction Processing Unit (TPU).

In this model, different types of complex transactions (CTs) are generated from the terminal with an arrival rate of $\lambda$. The activities of a complex transaction within the system are explained below.

When a transaction enters the system, it first visits the transaction manager and asks for permission to enter into the TPU. The TM allows a CT to enter into the TPU, if the number of CTs currently being executed in the TPU is less than the multiprogramming level ($N$). Once the TM allows a CT to enter into the TPU, the CT enters into the LMQ to access locks for its first base transaction (BT) using incremental static locking technique [44]. It is assumed that data items are accessed uniformly and independently by the BTs. If all the requested locks for a BT has been acquired by the lock manager, the transaction proceeds to one of the free query servers and become an active transaction; otherwise, it enters into the LMQ again to acquire the rest of the locks for the BT and becomes a blocked transaction. In our implementation, we have restricted to first come first serve queuing policy. A blocked transaction is again processed by the lock manager when its turn appears according...
to the queuing discipline followed in the LMQ. If at that time the transaction gets all of the requested locks, it becomes an active transaction.

When an active BT of a CT finishes its execution, the fate of the CT is decided by the fault diagnosis center. The following cases may arise. The active BT may be committing and thereby releasing all the locks; it goes out of the system if it is last BT of the CT. This case is full commit of the CT. After the commitment of the current BT, the CT may fire another BT and thus enters the LMQ again. The current BT may be aborted and the CT may have to be partially aborted, thus releasing all the locks of the BT. The CT returns back to the LMQ to fire the sequence of compensating base transactions (CBTs), i.e., CBTs which have to be fired in order to compensate the BTs fired after the last visited marked state (LVMS). Similarly, the current BT may be aborted and consequently CT may have to be fully aborted. Consequently, CT goes back to the LMQ to fire the sequence of CBTs as in the partial abort case but the CT leaves the system once the sequence of CBTs has been committed. Note here the CBTs are processed by the query servers like normal BTs but they are forced to commit.

Here, we assume that the number of query servers in the system is determined by the allowable multiprogramming level (N). Thus, whenever a CT is allowed to enter into the TPU one of the query servers will be available for it, if it can acquire the required locks. This has been done to see the effect of cooperation of the CTs without constraining the computing resources. Also we assume here that a CT may continue firing the next BT after commit of the current BT without any user waiting time (e.g. for input from the user who fired the CT at the terminal). The behavior of the users, which invoked the CT, is not taken into account in the simulation model.

7. Simulation parameters

In the simulation, we have used following parameters:

- **Full abort rate** ($\mu$): The full abort rate defines the probability that when a base transaction aborts, the complex transaction fully aborts. From now onwards, whenever we talk of ‘abort’, we will mean ‘full abort’ only.
- **Partial abort rate** ($\gamma$): The partial abort rate $\gamma$ defines the probability that when a base transaction aborts, the complex transaction aborts partially.
- **Full commit rate** ($\beta$): The full commit rate $\beta$ defines the probability that on successful completion of the base transaction, the complex transaction commits fully.
• **Arrival rate** (\( \lambda \)):
  This parameter indicates the rate of arrival of complex transactions from the terminals. The arrival pattern is assumed to have Poisson distribution.

• **Number of states** (\( k \)):
  This parameter denotes the number of ‘non-final’ states in the state transition graph of a complex transaction type.

• **Number of complex transaction types** (\( m \)):
  This parameter defines total number of complex transaction types. A complex transaction is a fired instance of a transaction type.

• **Cooperation rate** (\( \theta \)):
  This parameter defines the number of complex transaction types \( \theta m \) that can cooperate at a state \( s \) of a complex transaction type out of total \( m \) number of complex transaction types.

• **Marked state rate** (\( a \)):
  The marked state rate defines total number of \( ak \) marked states in the state transition graph of a complex transaction type out of total \( k \) ‘non-final’ states.

• **Write lock percentage** (\( W \)):
  This parameter defines the percentage of write locks out of total number of locks to be acquired by base transactions. If \( W \) is 1.0, all the locks will be acquired in write mode by the base transactions. When \( W \) is 0.0, all the locks will be acquired in read mode by the base transaction. Therefore, by adjusting this parameter in the range from 0.0 to 1.0, we can change the percentage of locks to be acquired in the write mode.

• **Multiprogramming level** (\( N \)):
  This parameter denotes maximum number of complex transactions allowed in the transaction processing system. Thus at any instant of time, a maximum \( N \) complex transaction may be there in the TPU, i.e., sum of number of complex transaction in lock waiting queue and number of complex transaction in the set of query servers. The system is said to be under-loaded if actual number of complex transactions in the TPU is less than multiprogramming level.

• **Number of database object** (\( D \)):
  This parameter denotes the number of distinct database objects present in the database. In the simulation, we have assumed that the database objects are locked uniformly and independently by the transactions.

• **Number of locks to be acquired by base transaction** (\( d \)):
  This parameter denotes the number of database objects to be acquired by a base transaction. This means that after acquiring locks on \( d \) database objects, a base transaction (BT) or compensating base transaction (CBT) may enter into the set of query servers.

• **Lock acquire time** (\( t \)):
  This parameter denotes the amount of the time lapsed in acquiring a lock i.e., lock manager consumes \( t \) unit of time to acquire a lock on the data object.

• **Base transaction computation time** (\( T \)):
  This parameter indicates the amount of the time a base transaction (BT) or a compensating base transaction (CBT) will be in the set of query servers. Thus, a \( T \) unit of time is needed for the query computation, by a query server.

• **Time out deadlock** (\( D \)):
  This parameter is used in deadlock removal policy. In the simulation, we have used timeout strategy to handle problem of deadlocks. This parameter indicates that the lock manager can wait up to maximum \( D \) time units to acquire a particular lock on the data object. If the lock requested by a base transaction is not granted within \( D \) time units by the lock manager, the base transaction is assumed to be in deadlock and it is forced to abort.

8. **The failure recovery model of a complex transaction**

We now present the fault model used to simulate a complex transaction. The complex transaction types are generated according to this fault model. In particular, the STG of different complex transaction types are generated by selecting suitable values of the following parameters:
• The total number of complex transaction types \((m)\).
• Number of ‘non-final’ states in the STG \((k)\); these non-final states are in addition to the two designated final states \(s_{\text{succ}}\) and \(s_{\text{fail}}\) which correspond to successful and failed (full abort) execution of the complex transaction.
• The cooperation rate \((\theta)\) \([0 \leq \theta \leq 1]\).
• The marked state rate \((\alpha)\) \([0 \leq \alpha \leq 1]\).
• The partial abort rate \((\gamma)\) \([0 \leq \gamma \leq 1]\).
• The full abort rate \((\mu)\) \([0 \leq \mu \leq 1]\).
• The full commit rate \((\beta)\) \([0 \leq \beta \leq 1]\).
• The multiprogramming level \((N)\). The multiprogramming level \(N\) is varied from a very low multiprogramming level to a very high multiprogramming level.
• The total number of database objects \((D)\).
• The total number of locks acquired by a base transaction \((d)\).

A generated complex transaction type may have total \(k + 2\) number of states including the successful and failure (abort) final states. The state \(s_1\) is assumed the start state of the complex transaction types. The cooperation rate \(\theta\) determines how many number of CT types may cooperate at any state \(s_i\). This is distributed randomly among the total \(m\) number of complex transaction types, so that out of total \(m\) complex transaction types, \(\theta m\) complex transaction types will be cooperating at each state. The marked state rate \((\alpha)\) determines the total number of marked states among the \(k\) states. This is also distributed randomly among the \(k\) states for a complex transaction type, so that \(\alpha k\) number of states will be marked out of total \(k\) states.

From a given state \(s_i\), a state transition leads to one of the states \(s_j[i,j \in \{1,\ldots,k\}\), \(s_{\text{succ}}\) and \(s_{\text{fail}}\) according to the scheme diagrammatically in Fig. 8. If a complex transaction is in state \(s_i\), then it may go to the state \(s_j\) either by commit of a fired BT or by abort of a fired BT. Also from state \(s_i\), it may go to one of the final states \(s_{\text{succ}}\) or \(s_{\text{fail}}\).

When the CT is in state \(s_i\), upon failure of the currently active BT in \(s_i\), it may take one of the following two actions: (a) it may choose to full abort the whole CT with a probability \(\mu\) and go to the ‘failure final state’ \(s_{\text{fail}}\) after firing the CBTs in the compensation process, or (b) it may partially abort the CT firing the CBTs in the compensation process and go to one of the states \(s_j, j \in \{1,\ldots,k\}\) with probability \(\gamma(i,j)\). Since the total probability for a CT to partially abort at state \(s_i\) is \(\gamma\), we take \(\gamma_{ij} = \gamma / k\), \(\forall i = \{1,\ldots,k\}\). The number of CBTs (fired during the compensation process) depends on the number of BTs that have been fired.

![Fig. 8. The state transition pattern of a complex transaction type.](image-url)
after the last visited marked state. After a partial abort to state $s_j$, the CT may again continue its computation from state $s_j$.

When a CT is in state $s_i$, upon commit of its currently active BT, it may take one of the following two actions: (a) it may choose to fully commit the whole CT with a probability $\beta$ and go to the 'successful final state' $s_{\text{succ}}$, or (b) it may go to one of the states $s_j, j \in \{1, \ldots, k\}$ with probability $c_{i,j}$. Therefore, the total commit probability of a BT in a state $s_i$ is

$$c_i^+ = \sum_{j=1}^{k} c_{i,j} = 1 - (\gamma + \beta + \mu)$$

8.1. State transition matrix of a complex transaction type

The state transition matrix of a complex transaction type is a probability matrix, which determines the execution sequences of a complex transaction type. A state transition matrix has $k$-rows (corresponding to each of the non-final states) and $2k + 2$ columns. The last two columns correspond to the failure and success states. For each non-final state $s_j$, we associate two columns $s_{sc}^j$ and $s_{sa}^j$ to differentiate the cases when the CT reaches the state $s_j$ from the state $s_i$ by commit or abort of the fired base transaction.

A state transition matrix is prepared for each of the $m$ complex transaction types in the transaction processing system. A typical state transition matrix of a complex transaction type (say $\text{CTType}_1$) is shown in the following table. A complex transaction goes to the failure state $s_{\text{fail}}$ (from state $s_i$) on the abort of its base transaction with a probability $c_i, j = c_i/k$, i.e., a complex transaction reaches to any of the non-final state due to the abort of the fired BT with equal probability. This event is thus related to the partial abort of the complex transaction. In the present simulation study, we have assumed that the probability of partial abort at each state $s_i$ is same.

Similarly, we also assume that the probability of full abort (and also full commit) of a complex transaction is same at all the states. In the state transition matrix the following conditions must hold for each row (i.e., for each state $s_i$)

- $\text{Prob}(\text{BT commit and firing a next BT}) = \sum_{i=1}^{k} s_{sc}^i = 1 - (\gamma + \beta + \mu)$
- $\text{Prob}(\text{Partial abort at state } s_i) = \sum_{i=1}^{m} s_{sa}^i = \sum_{i=1}^{k} (\gamma/k) = \gamma = \text{Partial Abort rate}$
- $\text{Prob}(\text{Full abort at state } s_i) = \mu = \text{Full Abort Rate}$
- $\text{Prob}(\text{Full commit at state } s_i) = \beta = \text{Full Commit Rate}$

<table>
<thead>
<tr>
<th>States</th>
<th>$S_1^i$</th>
<th>$S_2^i$</th>
<th>$S_3^i$</th>
<th>$S_4^i$</th>
<th>$S_5^i$</th>
<th>$S_6^i$</th>
<th>$S_7^i$</th>
<th>$S_8^i$</th>
<th>$S_{\text{fail}}$</th>
<th>$S_{\text{succ}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\mu$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\mu$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>...</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\mu$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>$S_i$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\mu$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>...</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\mu$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>$S_k$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\gamma / m$</td>
<td>$\mu$</td>
<td>$\beta$</td>
</tr>
</tbody>
</table>

9. Simulation results

Based on the queuing model and the fault model described in Section 8, we have conducted a series of simulation studies. The objective of the simulation studies is to examine the impact of the failure on the performance of a cooperative transaction. As discussed earlier, long-lived transactions suffer from overhead of the rollback recovery during system failures. The factors that can affect the amount of overhead are multiprogramming level ($N$), partial abort rate ($\gamma$), number of states of a complex transaction ($k$), number of locks held by a base transaction ($d$), and cooperation rate ($\theta$). In our simulation studies, we have varied the simulation parameters as follows:
• Multiprogramming level from light $N = 5$ to heavy $N = 25$.
• Cooperation rate from a minimum of $\theta = 0.0$ to maximum cooperation rate $\theta = 1.0$.
• Partial abort rate from a minimum of $\gamma = 0.0$ i.e., no rollback to high rollback i.e., $\gamma = 0.2$.
• Number of states of a complex transaction from a minimum $k = 5$ to maximum $k = 25$.
• Number of locks held by a base transaction from minimum $d = 2$ to maximum $d = 10$.

In the following table, we provide the exact value of the simulation parameters that are used in our simulation experiments. The values of the performance parameters are similar to those in previous performance studies [45,18]. The individual values of the variables that are changed during the experiments are noted separately in the figures.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full abort rate ($\mu$)</td>
<td>0.001</td>
</tr>
<tr>
<td>Partial abort rate ($\gamma$)</td>
<td>0.01</td>
</tr>
<tr>
<td>Full commit rate ($\beta$)</td>
<td>0.05</td>
</tr>
<tr>
<td>Arrival rate ($\lambda$)</td>
<td>0.1</td>
</tr>
<tr>
<td>Number of states ($k$)</td>
<td>10</td>
</tr>
<tr>
<td>Number of complex transaction types ($m$)</td>
<td>10</td>
</tr>
<tr>
<td>Cooperation rate ($\theta$)</td>
<td>0.5</td>
</tr>
<tr>
<td>Marked state rate ($z$)</td>
<td>0.5</td>
</tr>
<tr>
<td>Write lock percentage ($W$)</td>
<td>0.9</td>
</tr>
<tr>
<td>Multiprogramming level ($N$)</td>
<td>10</td>
</tr>
<tr>
<td>Number of database object ($D$)</td>
<td>1000</td>
</tr>
<tr>
<td>Number of locks in base transaction ($d$)</td>
<td>8</td>
</tr>
<tr>
<td>Lock acquire time ($t$)</td>
<td>0.1</td>
</tr>
<tr>
<td>Base transaction computation time ($T$)</td>
<td>10</td>
</tr>
<tr>
<td>Time out deadlock ($\varnothing$)</td>
<td>1000</td>
</tr>
</tbody>
</table>

The simulation was carried out with an event driven simulator. In the following, we discuss about different performance metrics used in the simulation.

9.1. Performance metrics

To compare fault impact on the transaction processing system, we have chosen following performance metrics.

• **CBT/DT ratio:** This parameter is defined as

  \[
  \text{CBT/DT ratio} (CT_i) = \frac{\text{Number of CBT fired by } CT_i}{\text{Number of BT fired by } CT_i}
  \]

  Thus, this parameter reflects the abort rate of the base transactions or the degree of compensation in the complex transaction $CT_i$.

• **Average saga length:** Saga length of a complex transaction $CT_i$ is defined as the number of the base transactions and compensating base transactions fired by the complex transaction

  \[
  \text{Saga length} (CT_i) = \text{Number of BTs of } CT_i + \text{Number of CBTs of } CT_i
  \]

• **Average service time per base transaction:** The service time of a base transaction is the time spent by a base transaction in the transaction processing system (TPU). The service time includes the time lapsed in the lock waiting queue, the time spent for computation in the query servers and the time spent in the process of acquiring the locks. Since a complex transaction fires multiple base transactions, therefore we take an average of the service times of base transactions being executed by a complex transaction. The average service time of the base transactions of a complex transaction is derived from the service time of the complex transaction and its saga length, i.e.,

  \[
  \text{Average service time per BT} = \frac{\text{Service time of } CT_i}{\text{Saga Length}}
  \]
We have conducted a series of experiments to analyze the effect of variation in simulation parameters on the performance metrics. We present these results and analyze them in the next subsections.

9.2. Effect of cooperation rate on system performance

We conducted the experiments to see the effect of cooperation rate on the cooperative transactions with different multiprogramming level and with different partial abort rate.

9.2.1. At different multiprogramming levels

First, we discuss the set of experiments to see the effect of multiprogramming level on the cooperative transactions.

9.2.1.1. Observations.

- Fig. 9 shows the effect of cooperation rate on the service time for base transaction with different multiprogramming levels. The average service time for the base transactions decreases with an increase on cooperation rate for different multiprogramming levels.
- Fig. 10 shows the effect of cooperation rate on the CBT/BT ratio. The CBT/BT ratio decreases with an increase in the cooperation rate for different multiprogramming levels.
- Fig. 11 shows the effect of cooperation rate on the average saga length of the complex transactions with different multiprogramming levels. It is evident that average saga length of the complex transactions increases with an increase in the cooperation rate.

9.2.1.2. Explanation. The cooperation rate parameter affects the service time of a complex transaction. Since at higher cooperation rate a base transaction obtains the locks earlier for a given a multiprogramming level, the effective service time of the base transaction is reduced because of shorter waiting time in the lock waiting queue (LWQ). The improvement in the service time at higher cooperation rate is shown in Fig. 9. For the same cooperation rate, a higher service time is obtained for higher multiprogramming level (Fig. 9). At higher multiprogramming level, there will be more complex transactions waiting in the LWQ due to conflicts while acquiring locks. Thus, the service time will be also higher for the higher multiprogramming level, even with same cooperation rate.

With higher cooperation rate, degree of interleaving among transactions would also be higher. Hence, a transaction can get its lock earlier and the frequency of deadlock would be less. This implies that the number of base transactions getting aborted would be less, i.e., the CBT/BT ratio decreases, as evident from Fig. 10. Thus at higher cooperation rate, as the number of base transactions being successfully executed increases, a higher saga length is obtained (Fig. 11). Again, with the same cooperation rate, a lower saga length and a higher CBT/BT ratio is obtained for higher multiprogramming level, as shown in Fig. 11. This is because at higher multiprogramming level, there will be more conflict among the complex transactions and thus more conflicts are resolved earlier.

![Av. Service Time Per BT Vs. COOPERATION RATE](image)

Fig. 9. Average service time per BT (with varying multiprogramming level).
number of complex transactions are likely to be fully aborted (because of deadlock). This would lead to lessen
the average saga length and increase the CBT/BT ratio.

9.2.2. At different partial abort rate
Now we discuss the set of experiments to see the effect of partial abort rate on the cooperative transactions.

9.2.2.1. Observations.
- Fig. 12 shows the effect of cooperation rate on the service time for base transaction with different partial
  abort rates. The average service time for the base transactions decreases with an increase on cooperation
  rate for different partial abort rates.
- Fig. 13 shows the effect of cooperation rate on the CBT/BT ratio with different partial abort rate. The CBT/
  BT ratio decreases with an increase in the cooperation rate for different partial abort rates. The rate of
decrease in the CBT/BT ratio is higher for lower partial abort rate. For higher partial abort rate, the
  CBT/BT ratio decreases more slowly.
- Fig. 14 shows the effect of cooperation rate on the average saga length of the complex transactions for dif-
  ferent partial abort rates. It is evident that average saga length of the complex transactions increases with
  an increase in the cooperation rate for all the partial abort rates.

9.2.2.2. Explanation. The average service time decreases at similar rates for different multiprogramming levels
as the cooperation rate increases, i.e., it becomes independent of partial abort rate (Fig. 12). This is due to the
fact that the partial abort rate really does not affect the time required by a base transaction to acquire its locks
and to execute itself. CBT/BT ratio also decreases as the cooperation rate increases, but at higher partial abort
rate, the rate of decrease of CBT/BT ratio will decrease, since it will be mostly dominated by the higher partial
abort rate (Fig. 13). Even with higher partial abort rates, average saga length increases at higher cooperation rate due to increase in the number of base transactions being executed successfully (Fig. 14).

9.3. Sensitivity analysis

We have conducted a set of experiments to perform a sensitivity analysis of our experiments on key simulation parameters. The two parameters that are quite interesting and should have an effect on our experiments are:

(a) Number of locks to be acquired by base transaction \(d\), and
(b) Number of states in a complex transaction \(k\).

With higher number of locks to be acquired by a base transaction, the system will have more possibility of conflicts. Similarly, with higher number of states in a complex transaction, the complex transactions will be larger in size, therefore increasing the possibility of more conflicts. Thus, we wanted to see how sensitive our simulation experiments are with respect to these parameters.
9.3.1. Observations

- **Fig. 15** shows the effect of cooperation rate on the service time for base transaction with different number of locks to be acquired by a base transaction \((d)\). The simulation has been done from smaller number of locks per base transaction \((d = 2)\) to higher number of locks per base transaction \((d = 10)\). The average service time for the base transactions decreases with an increase in cooperation rate for different values of \(d\); but for any particular cooperation rate, with higher value of \(d\), the average service time per BT is also higher.

- **Fig. 16** shows the effect of cooperation rate on the CBT/BT ratio with different values of \(d\). The CBT/BT ratio decreases with an increase in the cooperation rate for different \(d\). After a certain cooperation rate, the CBT/BT ratio becomes almost constant.

- **Fig. 17** shows the effect of cooperation rate on the service time for base transactions with different number of states in a complex transaction \((k)\). The average service time for the base transactions decreases with an increase on cooperation rate for different \(k\). The average service time remains almost same for different \(k\); thus becoming independent of the number of states in a complex transaction.

- **Fig. 18** shows the effect of cooperation rate on the CBT/BT ratio with different \(k\). The CBT/BT ratio decreases with an increase in the cooperation rate for different \(k\) with very small change in values of CBT/BT ratio for different values of \(k\).

9.3.2. Explanation

As the number of locks to be acquired by the base transactions increase, the conflict in the system increases; thus more transactions wait in the lock manager queue; thereby increasing the average service time per BT. Even with higher value of \(d\), due to the effect of cooperation within transactions, the service time decreases as the cooperation among the transactions increases, as shown in **Fig. 15**. Similarly, the CBT/BT ratio...
decreases initially as the cooperation rate increases for all the different values of \( d \), and after that it becomes near about constant (Fig. 16). CBT/BT ratio increases with higher values of \( d \). As the number of locks (\( d \)) increases, the chances of deadlock will also increase. Since timeout strategy has been adopted for the deadlock removal, more number of partial and full aborts will occur. This in turn will increase the number of compensating transactions.

Thus, even with high rate of concurrency and conflict in the system, our algorithm performs better as the cooperation among the transaction increases. In addition, the trend of decreasing of the average service time per BT and the decrease of CBT/BT ratio persists, proving the stability of the algorithm on the number of locks to be acquired by a base transaction.

The number states in a complex transaction (i.e., \( k \)) does not really affect the average service time per base transaction and the CBT/BT ratio as shown in Figs. 17 and 18 respectively. This is due to the fact that even with high number of states in a complex transaction, all the intermediate states (i.e., excluding the final states) of a transaction can be classified as non-final states, and thus can be clubbed together. Thus as the number of states in the complex transactions increase, the average service time per base transaction is not affected by \( k \). In addition, as \( k \) increases the number of base transactions that can be fired will increase. However, as the number of states (\( k \)) increase, the transaction becomes longer and the chances of its partial/full abort increases. Thus, for a given partial abort rate, the average number of CBTs will increase. Therefore, there appears to be no significant variation of CBT/BT with \( k \). Thus, this proves even when the complex transactions are quite large in size, our algorithm performs better with higher cooperation rates.

10. Discussion of performance results for the application scenarios

In this section, we will provide a short discussion describing the application characteristics and application performance needs for each of the application scenarios mentioned in Section 4. We will discuss whether
cooperativeness among the applications of a given application type is enabling to achieve the performance needs of the applications under the given application characteristics. In this quest, we will be referring to the simulation parameters used and simulation results of Section 9.

In Fig. 19, we show the application characteristics and application performance needs for each of the application types discussed in Section 4. The three crucial parameters to define application characteristics are:

- System load ($N$),
- partial abort rate ($\gamma$),
- No. of locks acquired by a base transaction ($d$).

The most important application performance metrics for the different application types are:

- Average service time,
- CBT/BT ratio.

### 10.1. Performance analysis of mobile transaction in 3G service environment

Mobile transactions in 3G service environment have the following characteristics: (a) high system load due to the large number of users to the system, (b) large number of database objects need to be accessed by individual base transactions, and, (c) a high partial abort rate due to mobility and disconnectivity of the mobile terminals. The mobile transactions in 3G environment need to have lower average service time and a lower CBT/BT ratio, to avoid the higher churn rate of the customers, and to avail better service quality. The corresponding simulation parameters chosen for this application scenario are shown in Fig. 19.

We see in Fig. 9, that with a heavy loaded system ($N = 25$), the system provides a better average response time when the mobile transactions ensure more cooperation among them. In Figs. 10 and 11, we see that even

<table>
<thead>
<tr>
<th>Application scenarios</th>
<th>Application Characteristics</th>
<th>Application performance needs</th>
<th>Application performance needs served by high cooperation rate ($\theta$)</th>
<th>Referenced Performance Results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3G service environment</strong></td>
<td>Yes ($\geq 25$)</td>
<td>Yes ($\geq 0.2$)</td>
<td>Yes ($\geq 10$)</td>
<td>Fig. 9, 10, 11, 12, 13, 14, 15</td>
</tr>
<tr>
<td><strong>Multi-site web-services</strong></td>
<td>Yes ($\geq 25$)</td>
<td>Yes ($\geq 0.2$)</td>
<td>Yes ($\geq 10$)</td>
<td>Fig. 9, 10, 11, 12, 13, 14, 15</td>
</tr>
<tr>
<td><strong>Multi-site workflow</strong></td>
<td>Yes ($\geq 25$)</td>
<td>No ($\leq 0.0$)</td>
<td>Yes/No ($\geq 10/2$)</td>
<td>Fig. 9, 10, 11, 15</td>
</tr>
<tr>
<td><strong>Ubiquitous Computing Environment</strong></td>
<td>No ($\leq 5$)</td>
<td>No ($\leq 0.0$)</td>
<td>No ($\leq 2$)</td>
<td>Fig. 9, 10</td>
</tr>
<tr>
<td><strong>Batch applications</strong></td>
<td>No ($\leq 5$)</td>
<td>No ($\leq 0.0$)</td>
<td>Yes ($\geq 10$)</td>
<td>Fig. 9, 10, 12, 15</td>
</tr>
<tr>
<td><strong>Intelligent Network (IN) Feature Composition</strong></td>
<td>Yes ($\geq 25$)</td>
<td>Yes ($\geq 0.2$)</td>
<td>No ($\leq 2$)</td>
<td>Fig. 9, 10, 11, 12, 13, 14</td>
</tr>
</tbody>
</table>

Fig. 19. Performance results of the application scenarios and used simulation parameters.
with a heavy loaded system \((N = 25)\), we achieve a decrease in CBT/BT ratio and an increase in average saga length with higher cooperation rate. This signifies that we have less compensation in higher cooperation rate. When we have a higher partial abort rate \((\gamma = 0.2)\) (as it is typical in mobile transaction environment), we will get a better average response time as the mobile transaction cooperates among themselves more, as shown in Fig. 12. In Figs. 13 and 14, we see that even in a system with high partial abort rate \((\gamma = 0.2)\), we achieve a better CBT/BT ratio decreases and average saga length increases, as the cooperation rate increases. This signifies that we have less compensation with higher cooperation rate. With high number of locks to be acquired by a base transaction \((d = 10)\), the average service time per BT decreases, as the cooperation among the applications increase, as shown in Fig. 15.

### 10.2. Performance analysis of multi-site composite web-services across multiple resource domains

We see in Fig. 19 that the multi-site composite web-services have the same application characteristics and application performance needs as that of mobile transactions in 3G service environment. The corresponding simulation parameters chosen for this application scenario are shown in Fig. 19, which is same as that in mobile transactions in 3G service environment. Thus, cooperation among the multi-site web-services provide the same performance improvements like in 3G service environment, as discussed in Section 10.1.

### 10.3. Performance analysis of multi-site workflows and multidatabase transactions using cooperating transactions

The multi-site workflows have the characteristics of having a higher system load due to a large number of users of the system, but they do not have a high partial abort rate as that of mobile transactions environment. The multi-site workflow applications can have both high number of locks acquired by the base transactions, as well as, small number of locks acquired by base transactions, depending on the applications. The multi-site workflow needs a lower CBT/BT ratio, but they can survive without having a lower average service time.

We see in Fig. 9, that with a heavy loaded system \((N = 25)\), the system provides a better average response time when the applications ensure more cooperation among them. In Figs. 10 and 11, we see that even with a heavy loaded system \((N = 25)\), we achieve a lower CBT/BT ratio and a higher average saga length with higher cooperation rate. This signifies that we have less compensation in higher cooperation rate. When we have a lower partial abort rate \((\gamma = 0.01)\) (as it is typical in multi-site workflow environment), we will get a lower average response time as the workflow transactions cooperate among themselves more, as shown in Fig. 12. In Figs. 13 and 14, we see that with a system having no partial abort \((\gamma = 0.0)\), we achieve a very low CBT/BT ratio and high average saga length. This signifies that we have very little compensation with higher cooperation rate. As the partial abort rate decreases, the CBT/BT ratio also decreases in a high rate. With high number of locks to be acquired by a base transaction \((d = 10)\), the average service time per BT decreases, as the cooperation among the applications increase, as shown in Fig. 15. Even when there is no need for acquiring high number of locks by a base transaction in a multi-site workflow \((d = 2)\), the average service time per BT decreases with a higher cooperation rate (Fig. 15). In a long durational multi-site workflow environment the cooperation among the transactions will ensure that the locks are released quicker and thereby they can avail a better average service time.

### 10.4. Performance analysis of controlled resource cooperation among concurrently running batch applications

The batch applications have the characteristic of acquiring a high number of locks on the database objects (or message objects in case of MQ). The batch applications do not incorporate high system load, since the number of batch jobs that run concurrently on the same database is normally low in number compared to online transaction systems. Also the batch jobs normally do not have high partial abort rate. The corresponding simulation parameters chosen for evaluating performance of cooperative batch jobs are shown in Fig. 19. Even with these application characteristics, the batch jobs have a performance need of having lower average service time, so that the batch window time can be reduced, thereby, planned down-time of online systems can be reduced. In addition, batch systems need to have lower CBT/BT ratio, so that we do not need compensation for the batch jobs, which is quite costly affair, due to the high CPU and database (I/O) usage of batch jobs.
We see in Figs. 9–11, that even with a lower system load \((N = 5)\), as characterized in batch environment, if we provide higher cooperation among the batch applications, we achieve (a) lower service time, (b) lower CBT/BT ratio, and (c) a higher average saga length respectively. This confirms the suitability of our cooperative transaction management scheme even in batch application environment. With higher number of locks acquired by base transaction, as is typical in a batch scenario \((d = 10)\), the average service time for a batch transaction decreases as the cooperation rate among the batch jobs increase, as shown in Fig. 15.

We believe that the proposed cooperative transaction mechanism can also be used inside DBMS-provided batch utilities (e.g., in data loading/unloading, table reorganizations, transaction schema binding, table statistics generation utilities), which are long running and locks the DBMS internal objects (e.g. schema description tables, table-spaces, index-spaces, plans, packages) and makes these objects non-available to other online/batch user applications for a long time.

10.5. Performance analysis of feature composition in intelligent networks (IN) applications

The composite feature transactions in IN environment have the following characteristics:

- High system load due to a large number of users of the telecommunication system.
- A high partial abort rate due to (a) interaction of the end-user with the base features which may result in abort of the base feature, (b) interaction among the features, and (c) mobility and disconnectivity of the mobile terminals.

The composite feature transactions in IN environment needs to have (a) lower average service time to provide a better responsiveness to the end-users and (b) a lower CBT/BT ratio to avoid the higher churn rate of the customers and to avail better service quality. The corresponding simulation parameters chosen for this application scenario are shown in Fig. 19.

We see in Fig. 9, that with a heavy loaded system \((N = 25)\), the system provides a better average response time when the composite feature transactions ensure more cooperation among them. In Figs. 10 and 11, we see that even with a heavy loaded system \((N = 25)\), we achieve a decrease in CBT/BT ratio and an increase in average saga length with higher cooperation rate; this signifies that we have less compensation in higher cooperation rate even in high load. When we have a higher partial abort rate \((\gamma = 0.2)\) (as it is typical IN composite feature transactions), we will get a better average response time as the composite features cooperate among themselves more, as shown in Fig. 12. In Figs. 13 and 14, we see that even in a system with high partial abort rate \((\gamma = 0.2)\), CBT/BT ratio decreases and average saga length increases, as the cooperation rate among the composite feature transactions increase. This signifies that we have less compensation with higher cooperation rate.

10.6. Performance analysis of control flow cooperation in ubiquitous computing applications

The transactions in Ubiquitous Computing Environment (UCE) normally do not generate high-loads, has low partial abort rate, and has low number of DB-locks acquired by their base transactions. UCE transactions need to have lower average service time, and lower CBT/BT ratio (see Fig. 19).

We see in Figs. 9–11, that even with a lower system load \((N = 5)\), as characterized in UCE transactions, if we provide higher cooperation among the UCE applications, we achieve (a) lower service time, (b) lower CBT/BT ratio, and (c) a higher average saga length – which confirms the suitability of our cooperative transaction management scheme even in UCE application environment.

11. Conclusions and future work

In this paper, we have analyzed the performance of the 2L-QuadLock scheme in presence of rollbacks and aborts by simulation. One of the objectives of this study has been to determine the extent of performance gain of the system with transaction cooperation, even in the presence of full/partial aborts and compensating actions. Another objective of this paper has been to show the efficacy of the presented cooperative transaction management scheme in different emerging application scenarios (e.g. in 3G-service environment, ubiquitous
computing environment, feature composition in intelligent networks, multi-site and multi-domain web-services). Simulation experiments with different parameter values of partial abort rates, multiprogramming level, number of states of a complex transaction, number of locks of a base transaction, and cooperation rates have been carried out. The following are the major observations of the simulation:

- Cooperation rate affects the average saga length, average service time for a base transaction, and CBT/BT ratio. A higher cooperation rate allows more transactions to interleave. Thus, a base transaction gets the lock at an earlier stage. This results in a reduced service time for a base transaction, a higher saga length due to possibility of executing more base transaction, and, a reduced CBT/BT ratio due to less number of compensating base transaction execution.
- Partial abort rate affects the degree of compensation i.e., the CBT/BT ratio. Even with higher partial abort rates, the CBT/BT ratio and the average service time per base transaction decreases with increase in cooperation rate.
- The system behaves in the same manner as the number of locks to be acquired by the base transaction \( d \) and the number of states in a complex transaction \( k \) increases. This is evident from the fact even with large value of \( d \) and \( k \), the CBT/BT ratio and the average service time per base transaction decreases as the cooperation rate increases.

Thus, these simulation results demonstrate the performance gain in 2L-QuadLock scheme due to high cooperation rate even in presence of higher failure rates, resource contention, multiprogramming level and the number of states in a complex transaction. We have also discussed the simulation parameters, which are effective in the different emerging application scenarios and the corresponding performance results.

In the present simulation scenario, we assume that a CT may continue firing the next BT after commit of the current BT without any user waiting time (e.g. for input from the user who fired the CT at the terminal). The behavior of the users, which invoked the CT, is not taken into account in the simulation model. These issues need to be investigated in future work by adjusting the simulation model by taking care of variable duration in between firing of the BTs and also the behavior of the user firing the CTs. In addition, we have chosen to carry out the simulation experiments using incremental static locking technique, since in cooperative transaction execution environment the locks to be held on the objects are known in advance. Thus, this simulation results will be realistic for this particular class of transactions. A simulation study of cooperative transactions with dynamic locking would be also an interesting future work.

Acknowledgement

The authors would like to thank the anonymous reviewers for their constructive suggestions and guidance, which definitely improved the quality of the paper.

References


Prithwish Kangsabanik obtained his Bachelors in Computer Science & Technology from Bengal Engineering College, Shibpur, India. He obtained his Masters and Ph.D. in Computer Science & Engineering from Indian Institute of Technology, Kharagpur. He is presently working as an Information & Communication Technology Architect in UBS, Switzerland. He will be joining Swisscom, the market-leading telecommunication service provider of Switzerland, as a Solution Architect from Mar, 2007. He has worked in various Industrial and academic R&D projects. His research interests are in the area of high-available system design, mobile & pervasive computing, computer networks, telecommunication systems, and active database management systems. He has published extensively in International Journals and Conference in the area of DBMS kernel design, application modeling & concurrency control in active DBMS, object oriented fuzzy expert systems design, and mobile transactions models in 3G service environment. He is a Senior Member of IEEE and a professional member of ACM.

Divakar Singh Yadav obtained his B.Tech. from G B Pant University, Pantnagar, India and his M.Tech in Computer Science & information Technology from Indian Institute of Technology, Kharagpur, India. He is currently working towards his Ph.D. in Dependable Systems & Software Engineering group at University of Southampton, UK as a Commonwealth scholar.

Rajib Mall completed his Bachelor’s, Master’s, and Ph.D degrees, all from Indian Institute of Science, Bangalore. He is currently working as a Professor at the Department of Computer Science and Engineering at Indian Institute of Technology, Kharagpur, India. He researches in the areas of analysis and testing of object-oriented programs and active database management systems.

Arun kumar Majumdar obtained B. Tech, M. Tech and Ph.D. degrees in Applied Physics from the University of Calcutta in 1967, 1968 and 1973, respectively. He also obtained a Ph.D. degree in Electrical Engineering from the University of Florida, Gainesville, USA, in 1976. He is currently working as a Professor in Department of Computer Science & Engineering at Indian Institute of Technology, Kharagpur. His research interests are in the area of Data and Knowledge-based Systems, Multimedia Systems, Medical Informatics, VLSI Design Automation. He is a Fellow of Indian National Academy of Engineering and Institute of Engineers (India). He is also a Senior Member of IEEE(USA).