2-2008

A Reservation-Based Extended Transaction Protocol

Wenbing Zhao
Cleveland State University, w.zhao1@csuohio.edu

Louise E. Moser
University of California-Santa Barbara, moser@ece.ucsb.edu

P. Michale Melliar-Smith
University of California - Santa Barbara, pmms@ece.ucsb.edu

Original Citation

Wenbing, Z. Y., Moser, L. E., & Melliar-Smith, P. M. (n.d.). A Reservation-Based Extended Transaction Protocol. IEEE Transactions on Parallel and Distributed Systems, 19, 2, 188-203.

A Reservation-Based Extended Transaction Protocol

Wenbing Zhao, *Member*, *IEEE*, Louise E. Moser, *Member*, *IEEE*, and P.M. Melliar-Smith, *Member*, *IEEE*

Abstract—With the advent of the new generation of Internet-based technology, in particular, Web Services, the automation of business activities that are distributed across multiple enterprises becomes possible. Business activities are different from traditional transactions in that they are typically asynchronous, loosely coupled, and long running. Therefore, extended transaction protocols are needed to coordinate business activities that span multiple enterprises. Existing extended transaction protocols typically rely on compensating transactions to handle exceptional conditions. In this paper, we identify a number of issues with compensation-based extended transaction protocols and describe a reservation-based extended transaction protocol that addresses those issues. Moreover, we define a set of properties, analogous to the ACID properties of traditional transactions that are more appropriate for business activities that span multiple enterprises. In addition, we compare our reservation protocol with other extended transaction protocols for coordinating business activities and present performance analyses and results.

Index Terms—Business activity, continuous availability, extended transaction model, isolation, relaxed atomicity, reservation protocol, transaction processing, Web services.

1 Introduction

The automation of business activities that are distributed across multiple enterprises becomes possible with the advent of the new generation of Internet-based technology, in particular, Web Services. Business activities typically involve related tasks that are loosely coupled and carried out over a long period of time. The automation of business activities, with direct computer-to-computer interactions and without human involvement, can provide substantial speed improvements and cost reductions for distributed enterprise computing. However, such enterprise applications must operate with a high degree of availability and performance. The resolution of inconsistencies among the databases of multiple enterprises is difficult, expensive, time consuming, and error prone, much more so than the resolution of inconsistencies within the databases of a single enterprise. Problems in the operation of the application services can adversely affect the relationships between an enterprise and its customers, suppliers, and partners.

Many enterprise applications are programmed using the transaction processing programming paradigm and are executed in the context of commercial transaction processing systems. Such systems typically employ transactional locking and the two-phase commit protocol for distributed transactions, which involves a transaction coordinator and one or more participants [12]. Although the two-phase commit protocol works well for the coordination of transactions within a single enterprise, the use of the two-phase commit protocol in distributed transactions that span multiple enterprises unavoidably involves one enterprise locking a data record of another enterprise. If the transaction coordinator fails, the locking period might be too long for the enterprise to tolerate. Even if the transaction coordinator does not fail, resources might be locked longer than the enterprise is willing to accept. Therefore, in practice, distributed transactions based on the two-phase commit protocol are seldom used for business activities that span multiple enterprises. Instead, such business activities are based on extended transactions [30], where each task of the business activity is executed as a sequence of one or more local transactions, and when a business activity is rolled back, compensating transactions [7] are applied to offset the effects of the committed local transactions.

Compensating transactions can be difficult to design and program and even more difficult to test in a distributed Web Service environment, resulting in a rather high level of errors. Even if the compensating transactions are designed and programmed correctly, it is difficult to ensure that when they are needed, the compensating transactions are applied correctly (for example, the number of retries must be limited for practical purposes). The typical result of an error in the design, programming, or deployment of a compensating transaction is inconsistency in the databases, either within a single database or, more probably, between the databases of different enterprises. It is difficult and expensive to resolve such inconsistencies, particularly when the databases are spread across multiple enterprises, as is the intention of Web Services. We demonstrate that the use of compensating transactions results in a much higher probability of database inconsistency than does the reservation protocol. Indeed, for entirely reasonable parameters, the probability of inconsistency approaches unity when using compensating transactions.

- *W. Zhao is with the Department of Electrical and Computer Engineering, Cleveland State University, 2121 Euclid Ave., Cleveland, OH 44115. E-mail: wenbing@ieee.org.*
- *L.E. Moser and P.M. Melliar-Smith are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA 93106. E-mail: {moser, pmms}@ece.ucsb.edu.*

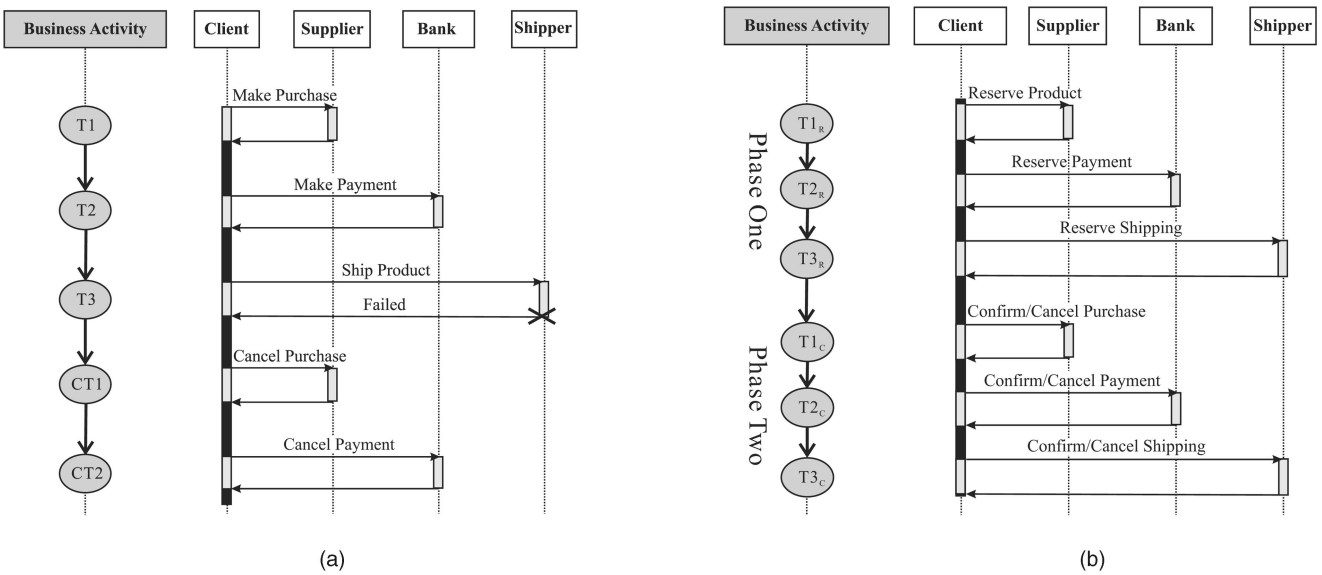

Fig. 1. Example business activity. (a) In the distributed transactions with compensating transactions approach, each task of the business activity is executed as a traditional ACID transaction within a single enterprise, with compensating transactions applied when a fault occurs. (b) In the reservation protocol approach, each task is executed as two subtasks, a reservation subtask and a confirmation/cancellation subtask; one of several possible interleavings of the tasks is shown in the figure.

The existing Web Services standards offer two alternatives:

- classical distributed transactions based on the two-phase commit protocol [4] and
- extended transactions with compensating transactions [5].

Apparently, the advocates of those strategies are unaware of the high probability of database inconsistency and the poor performance of those strategies.

In this paper, we describe a reservation-based extended transaction protocol that is compatible with and easily implemented using Web Services, that is easy to program and use, and that does not depend on compensating transactions. In our reservation protocol, except for read-only tasks, each task within a business activity is executed as two subtasks. The first subtask involves an explicit reservation of resources according to the business logic. The second subtask involves the confirmation or a cancellation of the reservation. Each subtask is executed as a separate traditional short-running ACID transaction. For example, the task of making a purchase of a certain amount of goods in a procurement business activity is executed as two subtasks. The first subtask reserves an amount of the goods to be purchased, and the second subtask converts the reservation into a purchase or a cancellation of the reservation. The reservation at the end of the first subtask becomes visible to other business activities, because fewer resources are available for them to reserve. However, this visibility does not compromise the isolation property, because the reservation can be confirmed or cancelled, and the other business activities cannot make any assumptions about resources that have not been reserved for them. For the duration of the reservation, the supplier grants an exclusive right to the client for the amount of goods reserved. During the second subtask, the reservation is confirmed only if the business activity can be completed successfully.

Fig. 1 shows an example business activity, highly simplified, of purchasing a product (resource) that requires shipping. In the compensating transactions approach shown in Fig. 1a, the buyer (that is, initiator) proceeds with the purchase only if both the product and the shipping are available. The buyer first places an order for the product and then arranges for shipping. However, if the shipper cannot deliver the product in time (for example, no trucks are available), the buyer must reverse the previous purchase, and then, a compensating transaction is applied. In the reservation protocol approach shown in Fig. 1b, each task is executed as two subtasks, a reservation subtask and a confirmation/cancellation subtask, without the need for compensating transactions. If the shipper cannot deliver the product in time, the buyer simply cancels the reservation of the product.

Despite the apparent similarity of our reservation protocol to the traditional two-phase commit protocol, there are significant differences, which we discuss later in this paper. In fact, our reservation protocol resembles more closely the escrow transactional method [23]. The similarity and the differences between our reservation protocol and the escrow transactional method are elaborated in depth. We also compare our reservation protocol with other extended transaction strategies. In addition, we present performance analyses and results, which indicate that the protocol performs similar to or better than other extended transaction strategies.

2 Model

A *transaction* is a set of operations on the application state that satisfies the following ACID properties [12]:

- *Atomicity.* Either all of the operations of the transaction succeed, in which case the transaction commits, or none of the operations is carried out, in which case the transaction aborts.

- *Consistency.* If the application state is consistent at the beginning of the transaction, the application state remains consistent after the transaction commits.
- *Isolation.* The transaction does not read or overwrite intermediate results produced by another transaction, that is, the transactions appear to execute serially.
- *Durability.* The updates to the application state become permanent (or persist) once the transaction is committed, even if a fault occurs.

A *local transaction* is a transaction with the above ACID properties that is executed at a single site.

A *task* is a short-duration unit of work that is executed as a sequence of one or more local transactions. A task can be modeled as an operation on one or more resources and typically modifies some of the attributes of those resources. For example, the task of purchasing/selling certain kinds of goods (application-defined resources) such as automobiles can be modeled as an operation in which the owner attribute of the resource is changed from the supplier to the buyer. An operation can be read-only, in which case the task is a *read-only task.* Examples of read-only tasks are tasks that obtain information such as a UPS or FEDEX shipping charge or a federal or state tax rate.

A *business activity* is a unit of work that consists of one or more tasks and that spans one or more enterprises, with messages sent between those enterprises. The tasks in a business activity are partially ordered. Tasks that are not causally related can be executed concurrently, and causally related tasks are executed according to the partial order. As in Fig. 1a, we use $T1$, $T2$, etc., to denote different tasks of a business activity. The corresponding compensation tasks are represented as $CT1$, $CT2$, etc. As in Fig. 1b, we use Ti_R to denote the reservation subtask for the ith task of the business activity, and Ti_C, the corresponding confirmation/cancellation subtask. A business activity has a number of participants; the participant that initiates the activity is called the *initiator.* The initiator is often the client of the other participants in the activity.

We assume that the participants in a business activity are subject to crash faults but not arbitrary (Byzantine) faults. We present more details on faults within the participants in a business activity and discuss how those faults are handled later as part of the description of the reservation protocol instantiation. Moreover, we assume that the communication between different participants in a business activity is reliable. An implementation of a specification such as the Web Services Reliable Messaging (WS-RM) specification [16] can be used to achieve reliable communication.

In a business activity, the ACID properties of a traditional transaction are not appropriate for the following reasons. First, it is not necessary that all of the participants see the same result, an observation that has been well recognized in practice. For example, if reservations are requested from several alternative participants, typically, one participant will see a confirmation, and the other participants will see a cancellation. The Web Services Business Activity (WS-BA) specification [5] defines two outcome types, *atomic outcome* and *mixed outcome*, to support different kinds of business activities, including those where only some of the participants agree on the same outcome. Thus, the failure of one task does not need to result in the rollback of the entire business activity. Moreover, the effect of executing a business activity might not be completely reversible, because of the business logic.

Analogous to a transaction that satisfies the ACID properties, a business activity satisfies a corresponding set of properties that we name the D4 properties and define below:

- *Distributed atomicity.* Each business activity defines a set of permissible outcomes and must reach one of those outcomes when it completes. These outcomes are classified as 1) *defined-commit outcomes*, for which the goal of the business activity is fulfilled either completely or partially, and 2) *defined-abort outcomes*, for which the goal of the business activity is not fulfilled.
- *Defined consistency.* At the end of each transaction within a business activity, the transaction must leave its data in a state that is not only locally consistent but also consistent with the data of other enterprises that are accessed by other business activities, which we refer to as *inter-enterprise consistency.* Both local consistency and inter-enterprise consistency are determined by the application logic, that is, business rules.
- *Disjoint resource isolation.* Disjoint sets of resources are assigned to concurrent business activities. A business activity can see whether its own requests for resources are granted or refused, but it cannot see the details of the requests of transactions within other concurrent business activities. Moreover, the ability of the transaction of a business activity to complete the processing of that business activity, within the resources that have been reserved for it, cannot be impeded by the transactions of other concurrent business activities.
- *Durability.* The results of a completed transaction within a business activity are persistent and cannot be undone accidentally by faults or other transactions.

3 The Reservation Protocol

In this section, first, we provide an abstract description of the reservation protocol. Next, we elaborate an instantiation of the reservation protocol, after presenting specific assumptions on which it is based. Finally, we establish informally the D4 properties for the reservation protocol instantiation.

3.1 Abstract Description of the Reservation Protocol

Consider a business activity involving n tasks $T1, T2, \ldots, Tn$ and m participants $P1, P2, \ldots, Pm$. Each task Ti involves k_i participants, $1 \leq k_i \leq m$. A participant may be involved in more than one task. The reservation protocol is executed in two phases, as described below.

During the first phase, for each task Ti, each of its k_i participants is contacted to execute the reservation subtask Ti_R. Let $Ti_R{}^j$ denote the reservation subtask executed by participant Pj. At the end of the first phase, the initiator decides the outcome of the business activity based on the results of the first phase and the business logic. A necessary but not sufficient condition for the completion of the business activity is that at least one participant has granted the reservation for each task, that is, for all Ti_R, there exists Pj such that $Ti_R{}^j$ is executed successfully with the reservation granted. If the decision is to proceed with the business activity, the initiator selects a set of subtasks, one for each task, to confirm the reservations granted.

During the second phase, for each subtask $Ti_R{}^j$ in the set, participant Pj is contacted to confirm the reservation

(denoted as confirmation subtask $Ti_C{}^j$), and the rest of the participants are contacted to cancel the reservation. If the decision is not to proceed with the business activity, all of the participants that have granted a reservation are contacted to cancel the reservations during the second phase.

Note that confirming a reservation is different from committing a transaction, and canceling a reservation is different from aborting a transaction. Read-only tasks do not require separate reservation and confirmation/cancellation subtasks.

3.2 Specific Assumptions for the Reservation Protocol Instantiation

We assume a flat business activity model; a hierarchical business activity model can be constructed based on the flat model. We assume that each task involves a single type of resource and that for each type of resource, one or more competing alternative participants are available to provide resources from which the initiator can choose. We assume that each subtask (that is, $Ti_R{}^j$ or $Ti_C{}^j$) is performed as a local transaction, that the result of the transaction, whether it is committed or aborted, is communicated to the invoker, and that the committed local transactions are durable.

As mentioned earlier, we assume that process crash faults are possible and that communication faults are not. A process crash fault might happen during either phase of the reservation protocol, and it might occur at a participant or at the initiator. If a participant fails after it has granted a reservation, we regard the failure as a second-phase failure (as far as the business activity is concerned). If a participant fails after it has processed the cancellation/confirmation request, it has no effect on the business activity. We assume that all faults are repaired in a timely manner to allow the fulfillment of business obligations.

The reservation protocol is driven by the initiator of a business activity. Most of the "transactional" logic that controls the progress and outcome of the business activity is in the initiator process. After each reservation or cancellation/confirmation subtask (that is, Ti_R or Ti_C), the initiator records the subtask and its result in a persistent log. When the business activity is completed, the logged records for that activity can be removed from the log (as far as the reservation protocol is concerned, though it would be good practice to preserve such records for other purposes). If the initiator recovers after it has failed in the middle of executing a protocol instance, it examines the logged records for all unfinished business activities and continues the protocol instances until they are completed.

We assume that the initiator is also protected by other appropriate high-availability mechanisms (see, for example, [3], [20], [32]) in order for it to recover quickly from faults, particularly during the second phase of the reservation protocol, so that it does not miss a deadline for a reservation that it placed during the first phase. If the initiator cannot satisfy this requirement, the D4 properties might not be satisfied.

We assume that all confirmation/cancellation messages are timestamped and are stored in a message log (either locally at or remotely from the initiator). In particular, on recovery from a fault, a participant can obtain such a message from the log. A participant determines the charge for the reservation based on the timestamp of the confirmation/cancellation message. A participant must honor a granted reservation, based on the timestamp of the message, even if the deadline (for a timed reservation) has passed at the time of recovery.

3.3 Instantiation of the Reservation Protocol

For each task, during the first phase of the reservation protocol, the reservation request is sent to the participants that provide the same service. The protocol proceeds forward only if at least one participant grants the reservation request for each task. Otherwise, the first phase is terminated and the initiator sends a cancellation request to each participant for which it granted a reservation. The business activity is then defined abort.

The reservation of a resource is categorized as one of the following two types:

- *Untimed reservation.* The resource is reserved indefinitely for as long as it takes.
- *Timed reservation.* The resource is reserved for a certain period of time. If the reservation deadline expires, the reservation is cancelled automatically by the resource holder.

The timed reservation has been a common practice in many businesses, for example, for airline seat and car rental reservations. In some cases, a surcharge is imposed by the resource holder if the reservation is not cancelled or confirmed by the deadline, for example, most hotels charge for a one-night stay if a reservation is not cancelled in time. We consider this surcharge to be part of the reservation cost. The timed reservation can lead to race conditions, for example, the resource owner might have timed out and cancelled the reservation by the time the user's confirmation message arrived. It is important for the user to confirm the reservation as early as possible to avoid this situation.

During the reservation period, the reserved resources are held exclusively for the client that made the reservation until the client confirms or cancels the reservation or the deadline has passed. In exchange for this service, the owner of the resources can charge the client a reservation fee. The reservation fee is either quoted by the resource owner or negotiated between the client and the resource owner prior to the reservation phase.

By explicitly reserving the resources involved in a task, the application has the flexibility of going forward or backward after the first subtask, without concern for the effects of the transaction, because a reservation allows either outcome. Even if other transactions see the intermediate result of the reservation (such as a reduction in the amount of available resources), those transactions cannot make any assumptions about the future of the resources that have not been reserved for them.

At the end of the first phase, the initiator determines which reservations to confirm and which to cancel, based on the application logic. For the subtasks to be executed during the second phase, the initiator sorts them based on their priorities and executes them in the following order:

1. confirmation of timed reservations,
2. confirmation of untimed reservations,
3. cancellation of timed reservations, and
4. cancellation of untimed reservations.

The subtasks in each of these four categories are sorted based on their priorities, and the subtasks are executed in the second phase according to that order.

The cancellation subtasks release any resources that have been reserved and, thus, make them available to the initiators of other reservations within the same or different business activities.

3.4 Maintenance of the D4 Properties by the Reservation Protocol Instantiation

In the following, we establish informally that the reservation protocol instantiation described above maintains the D4 properties, during both fault-free and fault conditions.

3.4.1 Distributed Atomicity

Under fault-free conditions, the reservation protocol instantiation maintains the distributed atomicity property, because each business activity defines its own set of permissible outcomes. Moreover, each reservation subtask and each confirmation/cancellation subtask is itself a transaction that satisfies the traditional ACID atomicity property.

Now, suppose that the initiator fails during the reservation phase. If the initiator can recover quickly enough to make reservations for all of the resources needed to complete the business activity, the outcome is defined commit. If the initiator cannot recover quickly enough to confirm some of the reservations before they expire, the initiator cancels all of its other reservations, and the outcome is defined abort.

If a participant fails and the initiator cannot reserve a resource from that participant, the initiator tries to find the resources from the alternative participants. If the initiator can find the resources from the alternative participants and successfully reserve them, a defined-commit outcome results unless the application logic mandates a defined-abort outcome.

If the initiator fails during the second phase and it can recover quickly enough to confirm or cancel all of the reservations, the failure has no effect on the outcome of the business activity and, thus, distributed atomicity is satisfied. Recall that we assumed that the initiator can recover promptly. If this assumption is violated and timed reservations are used in the business activity, the initiator is forced into a defined-abort outcome.

Now, suppose that a participant fails during the second phase and that as a result, the initiator cannot get immediate feedback from its confirmation or cancellation request for the reservation it placed with that participant. According to our assumptions, the participant must retrieve the confirmation/cancellation request from a message log and determine the reservation charge based on the timestamp included in the message. In particular, if a confirmation request is found with a timestamp before the deadline for a timed reservation, the request must be honored. Therefore, the failure of a participant during the second phase does not affect the outcome of the business activity. Consequently, the defined atomicity will be maintained.

3.4.2 Defined Consistency

The reservation protocol instantiation also maintains the defined consistency property of the business activity. The data are left in a consistent state, both locally within a single enterprise and distributed across multiple enterprises, because of the coordination of the reservation subtask and the confirmation/cancellation subtask and because the traditional ACID consistency property holds for each such subtask.

3.4.3 Disjoint Resource Isolation

Because the resources are held exclusively for the client that made the reservation, the reservation protocol instantiation ensures that disjoint sets of resources are assigned to concurrent business activities, regardless of faults. The protocol allows a business activity to see whether its own requests for resources are granted or denied within the reservation subtask, but it cannot see the details of requests or reservations by other concurrent business activities. This visibility does not compromise the disjoint resource isolation property, because the other business activities cannot make any assumptions about the future of the resources that have not been reserved for them.

Furthermore, the ability to complete the processing of a business activity, within the resources that have been reserved for it, cannot be impeded by the transactions of other concurrent business activities. When the cancellation subtask releases resources that have been reserved, those resources are made available to the initiators of other reservations within the same or different business activities. Thus, the resource protocol instantiation satisfies the disjoint resource isolation property.

3.4.4 Durability

Each transaction within a business activity satisfies the ACID durability property. In particular, the reservation subtask and the confirmation/cancellation subtask are transactions that satisfy the ACID durability property. This fact, together with the logging at the initiator after each subtask, ensures that the reservation protocol instantiation satisfies the durability property.

4 Comparison with Other Transactional Models

In this section, we provide an in-depth comparison of our reservation protocol with other transactional models. We start by comparing our reservation protocol with the escrow transactional method, because of the strong similarities between the two. We then compare our reservation protocol with other extended transaction models. We conclude this section by briefly comparing our reservation protocol with distributed transactions based on the two-phase commit protocol to avoid any confusion between the two.

4.1 Reservation Protocol versus Escrow Transactional Method

The escrow transactional method [23] allows concurrent updates from different transactions on some types of fields (typically aggregate fields) of the database records, and the reservation protocol also does so. The update operations are incremental or decremental in nature. The method involves some field quantities (referred to as resources in the reservation protocol), an application-supplied test of the quantity on hand (related to resource owners in the reservation protocol), and a final update of the field quantities (referred to as confirmation or cancellation in the reservation protocol). However, there are both fundamental and engineering differences between the escrow transactional method and the reservation protocol.

The escrow transactional method is intended for both long-running local transactions and distributed transactions. For distributed transactions, it uses the two-phase commit protocol, with the associated risk of long delays in the event of coordinator failure. In contrast, the reservation protocol involves the use of a reservation phase and a confirmation/cancellation phase to coordinate the tasks of a business activity across multiple enterprises.

In the escrow transactional method, once the database system has decided that the quantity-on-hand test is successful and has informed the application, it holds the escrow amount indefinitely until the application eventually performs the update operation and commits the transaction. The test-and-escrow operation is equivalent to the untimed reservation in the reservation protocol. However, real-world business rules typically do not allow the holding of resources indefinitely, especially if they are held for other enterprises. For practical applications, timed reservations must be supported. The reservation protocol specifies how the tasks are coordinated, and the redundancy level of the resources specifies how to achieve the D4 properties of the business activities.

Because of the context in which it was introduced, the escrow transactional method does not (and does need to) consider the recoverability of the applications that drive a long-running transaction (the application can simply abort all outstanding transactions on recovery). For the reservation protocol, this is not the case. The application that drives the reservation protocol must be made highly available, and all successful reservations must be logged, which implies that the failure of a local transaction can be accommodated by retrying that transaction or by selecting an alternative participant. For the escrow transactional method, the failure of a participant causes the entire distributed transaction to abort and be retried. Consequently, the risk of inconsistency for the escrow transactional method is similar to the risk of inconsistency for compensating transactions, shown in Fig. 4a.

The reservation protocol and the escrow transactional method also differ in their implementations. The escrow transactional method is implemented inside the database system as an additional mechanism to enable concurrent updates to some fields of the database records. The database system must maintain escrow records in an escrow journal. For this model to work, the application must indicate the test criteria and the desired update operations through SQL-like statements. For each escrow type field, the database system must maintain an extra data structure to store information needed for the recovery of the database system, such as a timestamp and the range of the quantity on hand (the lower limit is the original quantity minus the escrow amount, and the higher limit is the original quantity minus the committed amount). If the escrow test succeeds, the database system flushes the escrow record from the escrow journal.

The reservation strategy eliminates the need for the database system to be an expert on which fields are escrow types and to perform the associated escrow journaling tasks. In the reservation protocol, the application developers are responsible for determining which fields are escrow type fields and for introducing any additional fields that are necessary to perform the reservation and confirmation/cancellation subtasks. This approach is sensible, because the application developers know the data semantics best, not the database system, which is intended to carry out generic database services. The reservation protocol approach makes it unnecessary to introduce a proprietary extension to standard SQL, which the escrow transactional method requires.

4.2 Reservation Protocol versus Other Extended Transaction Models

Due to the limitations of traditional ACID transactions, other extended transaction models have been developed. Following the classification of Weikum [30], there are two major types of extended transaction models: 1) the *transactional workflow model* [1], [17], [31], which regards a business activity as a number of tasks executed as independent transactions, and 2) the *semantic transaction model*, which aims to preserve the ACID properties as much as possible while improving the performance of transactions by exploiting application semantics (for example, to allow relaxation of the atomicity and isolation properties).

The semantic transaction model can be further differentiated into two kinds of models: 1) the *transaction interleaving model* and 2) the *open nested transaction model.* The transaction interleaving model exploits the compatibility of different transactions to allow the concurrent execution of some transactions while ensuring the serializability of those transactions. In contrast, the open nested transaction model focuses on the compatibility of operations on abstract data types. Open nested transactions are naturally hierarchical. If an operation on a certain data type is open, a new sphere of control can be spawned. Such an operation is often executed as a subtransaction, and its result can be viewed by other subtransactions before the top-level transaction is committed. Operations within the subtransaction can be further mapped to lower-level subtransactions. The multilevel transaction model of Weikum [29] is a special case of the open nested transaction model in that its transaction tree is strictly layered. In contrast, the traditional multilevel transaction model of Lynch [19] can be regarded as closed nested transactions in which the sphere of control of a closed operation is its parent operation. A comprehensive discussion of nested transactions can be found in [22].

The well-known extended transaction protocol sagas [7] can be categorized as an open nested transaction strategy where the operations on the data types involved in a transaction are compatible with other transactions. However, sagas are also widely used in workflows as a means of coordinating different transactions within a business activity, in which case the operations on some data types might not be fully compatible with those of other transactions. Similarly, the reservation protocol can be regarded as a special open nested transaction strategy (specifically, a two-level open nested transaction strategy), and it can also be used in the context of workflows. However, there exist subtle differences between the reservation protocol and typical open nested transaction strategies. The reservation protocol always requires two phases of executions, whereas other open nested transaction protocols involve only a single phase if the transaction is successful. In a sense, the reservation protocol involves some loss of efficiency to gain the ease of cancellation of the reservation operations executed in the first phase.

All of the above extended transaction models rely on the use of compensating transactions to cancel (partially) the effects of some earlier committed transactions to ensure (partial) atomicity. For the semantic transaction model, this approach works well if the operations (or transactions) can be compensated. Unfortunately, that is not always the case. Some operations either cannot be compensated or are too expensive to compensate. In workflows, because the operations from different transactions on some data types cannot be guaranteed to be fully compatible, there is the additional risk of cascading compensations. These problems are well recognized in the literature [11], [28]. However, no effective solutions have been proposed to address them. For the

semantic transaction model, unless appropriate concurrency control is identified and enforced consistently to preserve the atomicity of a transaction, such operations must be deferred to commit time, which essentially forces the model to behave like the traditional transaction model [28]. In workflows, once such an operation is executed, the workflow must commit, which might lead to nonatomicity [11].

4.3 Reservation Protocol versus Transactional Locking and Two-Phase Commit

Distributed transactions based on transactional locking and two-phase commit involve both the locking of database records within a single enterprise and the two-phase commit protocol. At an abstract level, resource reservation and locking of a resource are similar in that, in both, the resource is put on hold temporarily. Furthermore, like the two-phase commit protocol, our reservation protocol involves two phases. Despite the apparent similarity of our reservation protocol to transactional locking and two-phase commit, there are a number of differences, which we discuss below.

In our protocol, the reservation of a resource is executed as a traditional ACID transaction. The application has full control over the reservation and how long the resource is reserved, whereas, in the two-phase commit protocol, the locking of a resource is internal to the database system and is transparent to the application, which has no control over how long the resource is locked. Note that this property is not unique to the reservation protocol—many other extended transaction protocols possess it as well.

Another difference between our reservation protocol and locking is the effect on other transactions that need to access the resource. If a resource is reserved and another transaction wants to access it, the transaction can acquire a lock on the resource, and the application can be informed immediately of the state of the resource (that is, some of the resource has been reserved, but a sufficient quantity of the resource remains to satisfy the reservation). Thus, the application can take an appropriate action without delay. However, if the resource is locked by the database system and another transaction wants to access it, the new transaction must wait until the lock is released. The waiting time might be long, in which case the application cannot take immediate action. Once again, this characteristic is not unique to the reservation protocol. The escrow transactional method also has this characteristic.

In the two-phase commit protocol, a fault at a participant might cause the rollback of the transaction, in which case that participant decides unilaterally to abort the transaction. In contrast, in our reservation protocol, only the initiator is authorized to commit or roll back the business activity. A fault at other participants might affect the initiator's decision and, thus, the outcome of the business activity; however, it does not necessarily result in the rollback of the entire business activity. Again, this characteristic is not unique to our reservation protocol. There exist other extended transaction protocols such as the business agreement protocols described in the WS-BA specification [5] that have this characteristic. However, they are based primarily on compensating transactions.

5 Availability and Consistency Analyses

For the purposes of the availability and consistency analyses, we model a business activity as a sequence of one or more tasks, each of which is a traditional ACID transaction. The concurrency of tasks within the business activity has no effect on our availability and consistency analyses.

We consider only faults that are detected immediately so that a transaction can be aborted and retried immediately. Detection might involve the operating system, the database, the transaction or communication middleware, and custom-coded application data validity checks. We do not consider faults that allow a transaction to appear to complete even though they yield incorrect results that were not detected.

Following the detection of a fault in a transaction, the transaction or the entire business activity is aborted and retried. We assume that the standard transaction commit and abort mechanisms operate correctly. We consider only a single retry of the business activity and a single retry of compensating transactions when they are used.

5.1 Architectures

For comparison and evaluation, we consider four architectures:

- *No Fault Recovery Architecture.* There are no attempts at recovery from faults.
- *Single Distributed Transaction Architecture.* A business activity is modeled as a single distributed transaction using the two-phase commit protocol. If any part of the business activity fails, the entire distributed transaction is aborted and the business activity is then retried once only.
- *Abort of Transactions with Compensating Transactions Architecture.* A business activity is still modeled as a single distributed transaction. If any subtransaction fails, every completed subtransaction is aborted by a compensating transaction, and the entire business activity is retried once only.
- *Reservation Protocol Architecture.* A business activity is modeled as a sequence of individual transactions that comprise pairs of reservation and confirmation/cancellation transactions. If a transaction within the business activity fails, the transaction is aborted and is retried individually, rather than the entire business activity being retried. We allow only one transaction within the business activity to be retried. This restriction is imposed to yield a fair comparison with the other architectures in which the single distributed transaction is retried once only.

We consider the effect that each of these architectures has on the availability of the business application and the consistency of the databases, particularly when it involves multiple enterprises. In particular:

- We investigate the probability that all or a large number of business activities will complete successfully. Many businesses must process thousands or millions of activities every day. Each activity that does not complete successfully can involve difficult and expensive manual intervention.
- We investigate the probability that the databases of the business activity might be left in an inconsistent state, an inconsistency that might spread across multiple enterprises. Even a potential inconsistency can require difficult and expensive manual intervention.

There is no intention that the calculations presented here provide accurate probabilities for any particular business application. They are intended only to investigate the effects

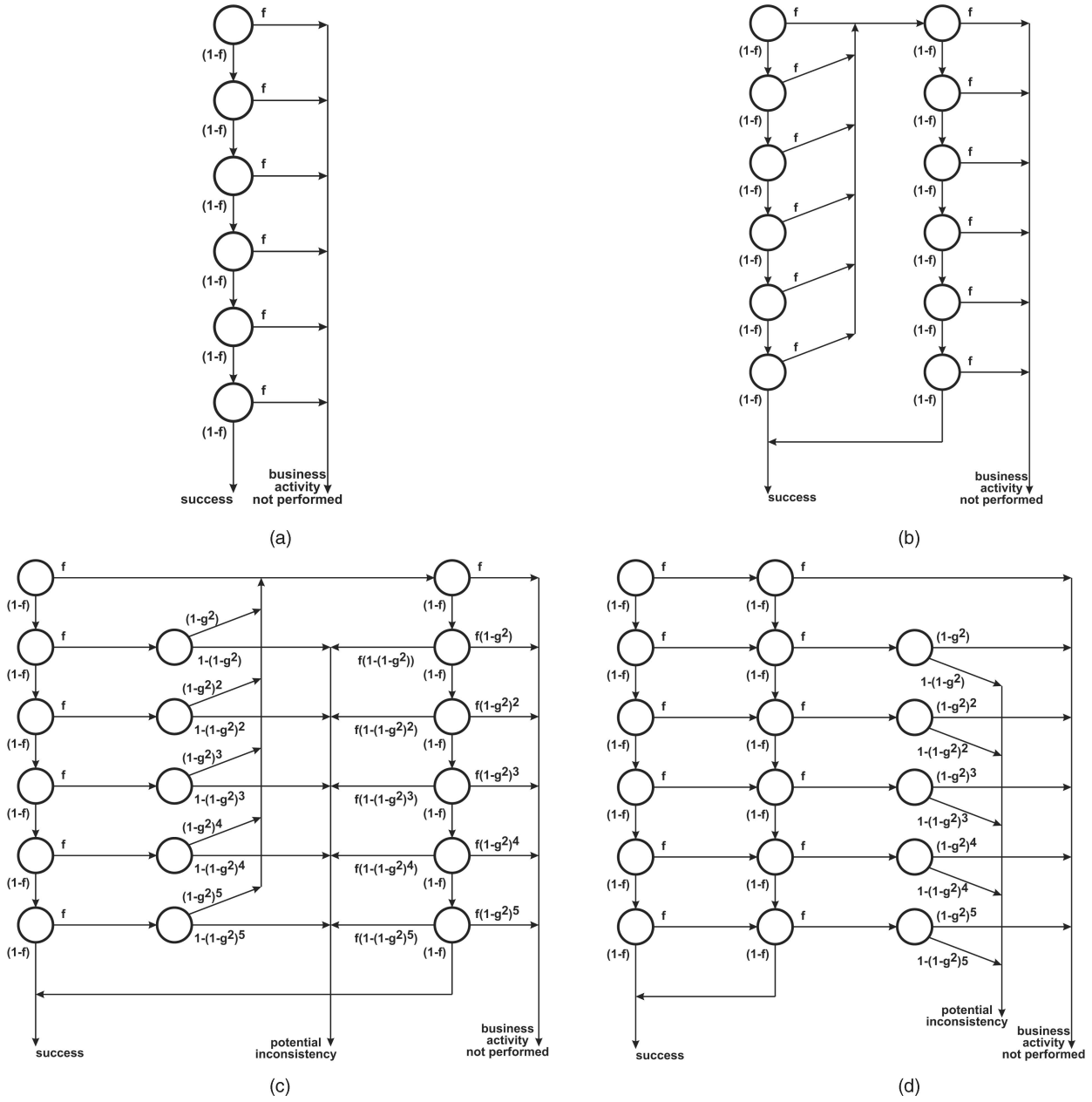

Fig. 2. The Markov models for the four architectures. (a) No Fault Recovery Architecture. (b) Single Distributed Transaction Architecture. (c) Compensating Transaction Architecture. (d) Reservation Protocol Architecture.

on availability and consistency for the particular architecture chosen.

5.2 Markov Models and Parameters

We estimate the probability that all of the transactions in a business activity complete successfully and the probability that the databases might have been left in an inconsistent state. We use a discrete-time Markov model for each architecture. The parameters of the Markov models are given as follows:

- n: the number of transactions in a business activity.
- m: the number of business activities over the period of interest (per hour, per day, etc.).
- f: the probability that a single transaction does not complete successfully, $0 < f < 1$.
- g: the probability that a compensating transaction does not complete successfully, $0 < g < 1$.

Thus, we assume that each transaction has the same probability f of failure and that each corresponding compensating transaction has the same probability g of failure.

In the illustrations of the Markov models, for convenience of presentation, we show only $n = 6$ transactions. The illustrations of the Markov models can be easily extended to more transactions. Real business activities typically involve more than six transactions, as do the business activities shown in our results below.

5.2.1 No Fault Recovery Architecture

Fig. 2a shows the Markov model for calculating the probability that a business activity with n transactions

completes successfully with no attempts at recovery from faults. This architecture is included as a baseline.

The probability that all of the m business activities complete successfully is given by

$$(1-f)^{nm}.$$

5.2.2 Single Distributed Transaction Architecture

Fig. 2b shows the Markov model for a business activity that is processed as a single distributed transaction using the two-phase commit protocol. If a subtransaction fails, the entire business activity is aborted and retried once only. We assume that the abort is correct and, thus, that there is no risk that the databases are left in an inconsistent state.

The probability of success on the first attempt of the business activity is $(1-f)^n$, and the probability of aborting the first attempt of the business activity is $1-(1-f)^n$. The probability of success of the retry of the business activity is $(1-f)^n$. The overall probability of success of the business activity is given by

$$(1-f)^n + (1-(1-f)^n) * (1-f)^n = (1-f)^n(2-(1-f)^n).$$

Thus, the probability that all of the m business activities complete successfully is given by

$$\Big((1-f)^n(2-(1-f)^n)\Big)^m.$$

5.2.3 Abort of Transactions with Compensating Transactions Architecture

Fig. 2c shows the Markov model for this architecture, where a business activity is still processed as a single transaction, but if a subtransaction fails, each completed subtransaction is aborted by a compensating transaction. The entire business activity is retried once only, and each compensating transaction is retried once only. If a compensating transaction cannot be completed successfully, even after a retry, the potential exists that databases are left in an inconsistent state and that manual intervention is required.

This model is also appropriate for the distributed escrow transactional method, although the value of g will be less because for the distributed escrow transactional method, the compensations are simple rather than arbitrarily complex as they might be for the general compensating transactions architecture.

The probability of success on the first attempt of the business activity is given by

$$(1-f)^n.$$

If a fault occurs in the second subtransaction and the first transaction is compensated, the probability that the first attempt of the compensation fails and the retry of that attempt fails is g^2, and the probability that either the first attempt of the compensation or the retry of that attempt succeeds is $1-g^2$. If a fault occurs in the third subtransaction, the first and second subtransactions are compensated. The probability that the compensations of both the first and second subtransactions succeed is $(1-g^2)^2$, and the probability that one or the other of those compensations does not succeed is $1-(1-g^2)^2$.

Thus, the probability that a fault occurs during the first attempt of the business activity so that the first attempt is aborted and the successful transactions of the first attempt are compensated successfully to allow the retry of the business activity is given by

$$\begin{aligned}
&f + (1-f)f(1-g^2) + (1-f)^2 f(1-g^2)^2 + \ldots + \leftarrow \\
&(1-f)^{n-1} f(1-g^2)^{n-1} \\
&= f\left(\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right),
\end{aligned}$$

using the formula for the sum of a finite geometric series.

The probability of success on either the first or the second attempt of the business activity is given by

$$\begin{aligned}
&(1-f)^n + (1-f)^n\left(f\left(\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right)\right) \\
&= (1-f)^n\left(1+f\left(\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right)\right).
\end{aligned}$$

Consequently, the probability that all of the m business activities complete successfully is given by

$$\left((1-f)^n\left(1+f\left(\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right)\right)\right)^m.$$

If a compensating transaction fails and the retry of the compensating transaction also fails, the databases are left in a potentially inconsistent state, and manual intervention is required.

The potential inconsistency might arise due to a fault in the first attempt of the business activity and the failure to compensate the first attempt with probability

$$\begin{aligned}
&(1-f)f(1-(1-g^2)) + (1-f)^2 f(1-(1-g^2)^2) + \ldots + \leftarrow \\
&\quad (1-f)^{n-1} f(1-(1-g^2)^{n-1})\leftarrow \\
&= f\Big(1 + (1-f) + (1-f)^2 + \ldots + (1-f)^{n-1}\Big) \\
&- f\Big(1 + (1-f)(1-g^2) + (1-f)^2(1-g^2)^2 + \ldots + \leftarrow \\
&\quad (1-f)^{n-1}(1-g^2)^{n-1}\Big) \\
&= f\left(\frac{1-(1-f)^n}{1-(1-f)\leftarrow} - \frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right),
\end{aligned}$$

using the sum of a finite geometric series twice.

The potential inconsistency might also arise due to a fault in the first attempt that is successfully compensated, followed by a retry of the entire business activity, during which a further fault occurs for which compensation is not successful. The probability that a fault occurs in the first attempt and is successfully compensated is given by

$$f\left(\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right).$$

The probability that a fault occurs in the second attempt and cannot be compensated is the same as the probability that a fault occurs in the first attempt and cannot be compensated. Thus, the probability of a potential inconsistency in either attempt is given by

$$
\begin{aligned}
&f\left(\frac{1-(1-f)^n}{1-(1-f)\leftarrow}-\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right)\\
&\quad+f\left(\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right)\\
&\quad\times f\left(\frac{1-(1-f)^n}{1-(1-f)\leftarrow}-\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right)\\
&=f\left(1+f\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right)\\
&\quad\times\left(\frac{1-(1-f)^n}{1-(1-f)\leftarrow}-\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right).
\end{aligned}
$$

Consequently, the probability that within m business activities, the databases are left in a potentially inconsistent state is given by

$$
1-\left(1-f\left(1+f\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right)\times\left(\frac{1-(1-f)^n}{1-(1-f)\leftarrow}-\frac{1-(1-f)^n(1-g^2)^n}{1-(1-f)(1-g^2)\leftarrow}\right)\right)^m.
$$

5.2.4 Reservation Protocol Architecture

Fig. 2d shows the Markov model for this architecture, where each transaction of the business activity is retried individually, rather than the entire business activity being retried. We assume that only one transaction in a business activity can be retried to yield a fair comparison with the distributed transaction architecture, where the entire distributed transaction is retried once only. If that retry also fails, the business activity fails, and all of its reservations are cancelled. Those cancellations can be either explicit cancellations or expirations of reservations. In both cases, the cancellation requires the execution of a transaction that might not complete, and the model includes the probability of such a fault. One retry is allowed for each explicit cancellation or expiration of a reservation.

The probability that the business activity completes successfully with no transactions failing is $(1-f)^n$. The probability that exactly one of n transactions failed and then succeeded when retried while the other $n-1$ transactions succeeded is $nf(1-f)^n$. Thus, the probability that a business activity completes successfully is given by

$$
(1-f)^n+nf(1-f)^n=(1-f)^n(1+nf).
$$

Consequently, the probability that all of the m business activities complete successfully is given by

$$
((1-f)^n(1+nf))^m.
$$

The probability that two transactions or one transaction and its retry have failed is given by

$$
f^2+2f^2(1-f)+3f^2(1-f)^2+\ldots+nf^2(1-f)^{n-1}.
$$

In this case, all successfully completed reservations must be cancelled. Two attempts are allowed for each cancellation transaction. The probability that one of the reservations is not cancelled (because the local transaction for a reservation cancellation failed even when retried) is given by

$$
\begin{aligned}
&2f^2(1-f)(1-(1-g^2))+3f^2(1-f)^2(1-(1-g^2)^2)+\ldots+\leftarrow\\
&nf^2(1-f)^{n-1}(1-(1-g^2)^{n-1}))\leftarrow\\
&=f^2\left(1+2(1-f)+3(1-f)^2+\ldots+n(1-f)^{n-1}\right)\\
&\quad-f^2\left(1+2(1-f)(1-g^2)+3(1-f)^2(1-g^2)^2+\ldots+\leftarrow\right.\\
&\quad\left.n(1-f)^{n-1}(1-g^2)^{n-1}\right)\\
&=f^2\ \frac{1-(n+1)(1-f)^n+n(1-f)^{n+1}}{(1-(1-f))^2}\Bigg)\Bigg(\\
&\quad-f^2\ \frac{1-(n+1)(1-f)^n(1-g^2)^n+n(1-f)^{n+1}(1-g^2)^{n+1}}{(1-(1-f)(1-g^2))^2}\Bigg),\Bigg(
\end{aligned}
$$

where we have applied the following formula twice:

$$
1+2x+3x^2+\ldots+nx^{n-1}=\frac{1-(n+1)x^n+nx^{n+1}}{(1-x)^2}.
$$

Consequently, the probability that for m business activities, the reservation cancellations succeed is given by

$$
\begin{aligned}
&1-f^2\ \frac{1-(n+1)(1-f)^n+n(1-f)^{n+1}}{(1-(1-f))^2}\Bigg)\Bigg(\\
&+f^2\left(\frac{1-(n+1)(1-f)^n(1-g^2)^n+n(1-f)^{n+1}(1-g^2)^{n+1}}{(1-(1-f)(1-g^2))^2}\right)\Bigg)^m
\end{aligned}
$$

and, thus, the probability that within m business activities, the databases are left in a potentially inconsistent state is given by

$$
\begin{aligned}
&1-\quad1-f^2\ \frac{1-(n+1)(1-f)^n+n(1-f)^{n+1}}{(1-(1-f))^2}\Bigg)\Bigg(\\
&+f^2\left(\frac{1-(n+1)(1-f)^n(1-g^2)^n+n(1-f)^{n+1}(1-g^2)^{n+1}}{(1-(1-f)(1-g^2))^2}\right)\Bigg)^m.
\end{aligned}
$$

5.3 Comparison of Availability and Consistency

For this comparison, we assume that there are $n=20$ transactions per business activity for the No Fault Recovery Architecture, the Single Distributed Transaction Architecture, and the Abort of Transactions with Compensating Transactions Architecture. For the Reservation Protocol Architecture, we assume that there are $n=40$ transactions per business activity, that is, 20 tasks, each of which involves two subtasks (reservation and confirmation/cancellation), each of which is a transaction.

In all of the graphs, the x-axis represents the number m (from 1 to 10^9) of business activities within an hour, day, week, month, year, or whatever time period is of interest. Each curve on the graph represents the probability (from 10^{-3} to 10^{-6}) that an individual transaction fails. Here, we assume that the probability g that a compensating transaction or a reservation cancellation (or expiration) transaction fails is the same as the probability f that a regular transaction fails.

Fig. 3a shows the probability that all of the m business activities complete successfully when there is no attempt to recover from faults. Note that because there is no attempt to

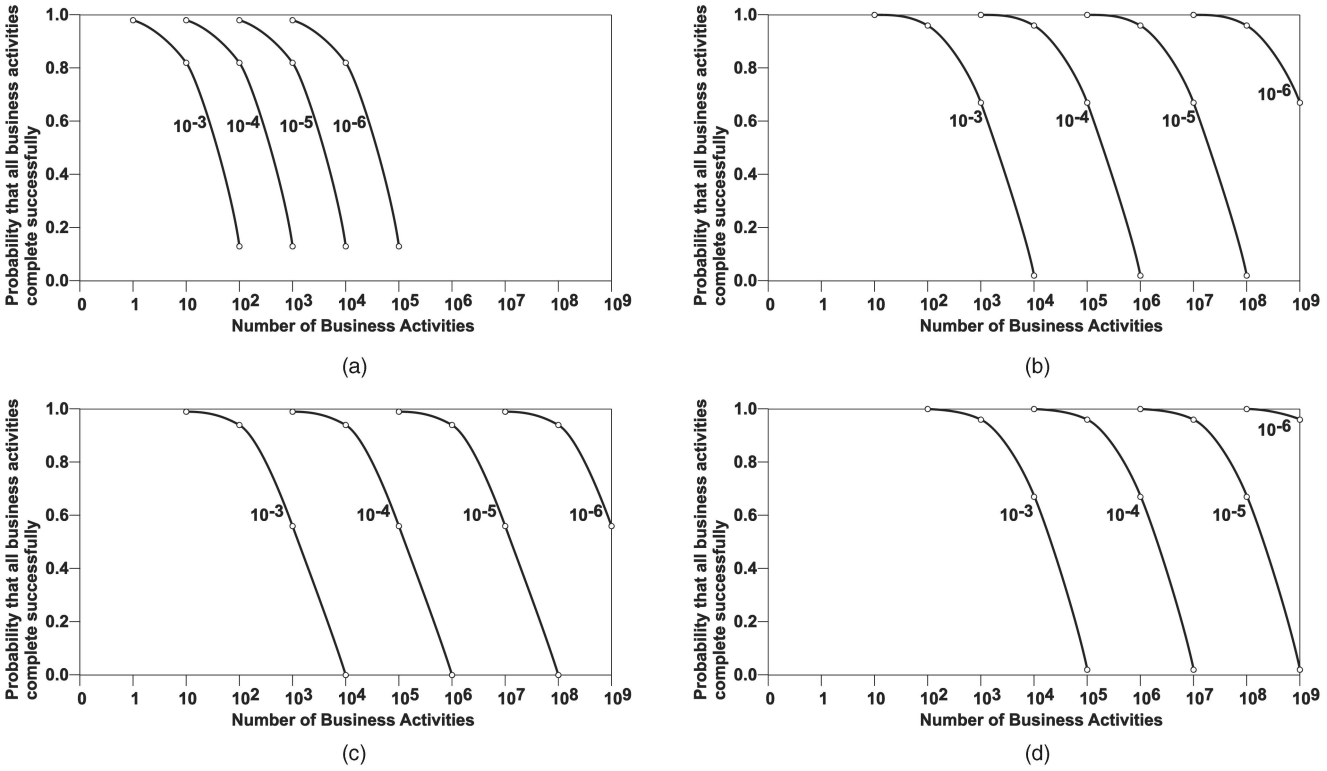

Fig. 3. The probability that all of the m business activities complete successfully for the different architectures. In the Abort of Transactions with Compensating Transactions Architecture, compensating transactions have the same fault rate as regular transactions. In the Reservation Protocol Architecture, reservation expiration transactions have the same fault rate as regular transactions. (a) No Fault Recovery Architecture. (b) Single Distributed Transaction Architecture. (c) Compensating Transactions Architecture. (d) Reservation Protocol Architecture.

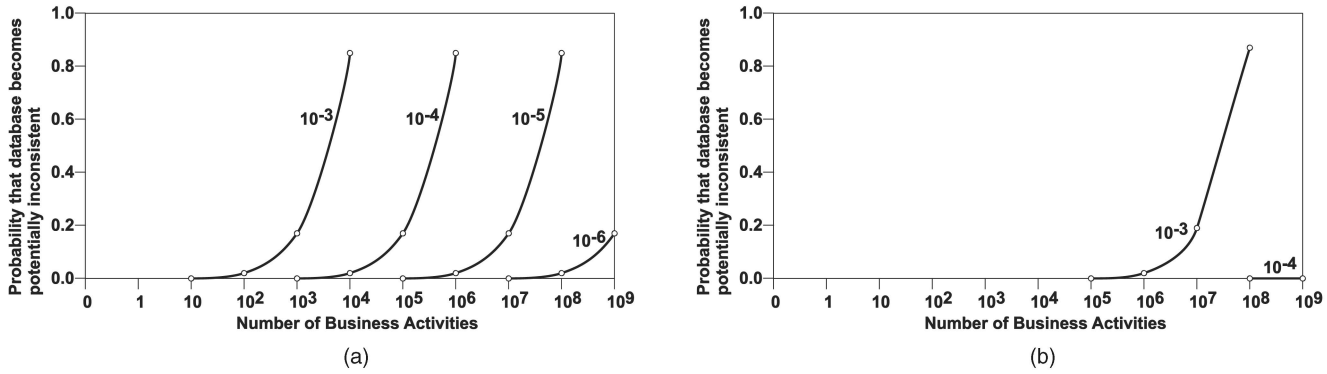

Fig. 4. The probability that the databases are left in a potentially inconsistent state after m business activities for the Compensating Transactions Architecture and the Reservation Protocol Architecture. (a) Compensating Transaction Architecture. (b) Reservation Protocol Architecture.

recover from faults, the probability of success deteriorates rapidly as more business activities are attempted.

Fig. 3b shows the probability that all of the m business activities complete successfully with the Single Distributed Transaction Architecture. Note that for larger numbers of business activities, even a single retry of a business activity yields much better probabilities of success. However, the Single Distributed Transaction Architecture is seldom employed because of the risk that the failure of the transaction coordinator in a computer of one enterprise might block transactions in the computers of other enterprises and might render inaccessible the data of those other enterprises. In current practice, typically, the Abort of Transactions with Compensating Transactions Architecture is used instead.

Fig. 3c shows the probability that all of the m business activities complete successfully with the Abort of Transactions with Compensating Transactions Architecture. The availability is good but not quite as good as that for the Single Distributed Transaction Architecture. However, this graph shows the most optimistic scenario, with compensating transactions assumed to have the same fault rate as regular transactions and with the retry of compensating transactions. Experience has shown that in practice, compensating transactions are difficult to program and that the fault rate for compensating transactions is much higher than the fault rate for regular transactions, because

compensation is an unanticipated activity that is difficult to plan for and difficult to test. Some enterprises do not attempt to retry transactions, but rather proceed directly to manual rectification if a compensating transaction fails.

Fig. 3d shows the availability achieved by the Reservation Protocol Architecture, where the number of transactions per business activity is increased from $n = 20$ to $n = 40$, because there are two subtasks (transactions) of each of the 20 tasks of the business activity. The comparison of the figures shows that the Reservation Protocol Architecture achieves better availability than the other architectures, although the improvement might not be sufficient to be decisive.

The probability of completing business activities is not the only important metric. The probability that the databases are left in a potentially inconsistent state is even more critical than the probability that business activities do not complete. Figs. 4a and 4b show the probabilities of potential inconsistency for the Abort of Transactions with Compensating Transactions Architecture (with $n = 20$) and the Reservation Protocol Architecture (with $n = 40$). Here, compensating transactions and reservation cancellation (or expiration) transactions are assumed to incur the same fault rate as regular transactions. The Reservation Protocol Architecture has superior performance because there are fewer additional transactions for each fault recovery. It is our assessment that the differences in the probabilities that the databases are left in a potentially inconsistent state presents a decisive advantage for the Reservation Protocol Architecture.

6 Concurrency and Latency Analyses

The most prominent advantage of the reservation protocol is that it provides the potential for increased concurrency. When a request is made to reserve a specific quantity of a resource, only that much of the resource is reserved, and the remainder of the resource continues to be available to other business activities. With distributed transactions based on transactional locking, the request for that specific quantity of the resource causes the locking of the database record for the available resource, and other business activities that need the same resource must wait until the business activity that locked the database record completes, as Fig. 5 shows. For example, requesting two seats on an airline flight causes all of the seats on that flight to be locked for the duration of the transaction, and other customers seeking seats must wait. Consequently, the reservation protocol permits a higher degree of concurrency than the distributed transaction protocol for business activities that involve the same resource.

The most important performance metric from the point of view of a client is the response time (latency) required to satisfy the client's request. A general analysis of the latency of a business activity is limited by the lack of information about actual execution times of the tasks within the business activity, the probabilities of alternative paths of tasks through the business activity, and the concurrency of tasks within the business activity. Even with this limitation, we can obtain interesting and significant results for the time to completion of a business activity.

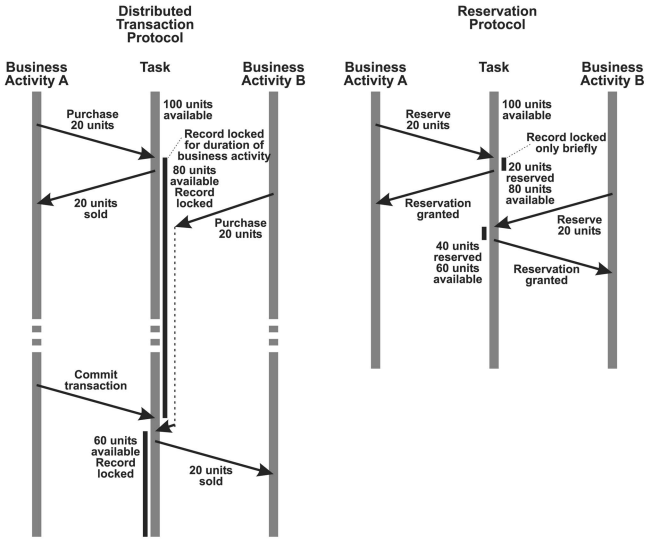

Fig. 5. Distributed transactions, based on transactional locking, lock database records and, thus, might delay other business activities that the reservation protocol does not delay.

The latency of a business activity is affected by two kinds of events:

- faults and recovery from faults and
- locking and blocking caused by locking.

The effects of faults on the latency of a business activity are either relatively easy to estimate if the recovery from a fault is successful or imponderable if inconsistencies occur. Furthermore, faults are sufficiently infrequent, and interactions between faults are rare and, thus, they do not have a significant effect on the latency. Consequently, we do not undertake an analysis of the effects of faults on the latency of a business activity here.

The effects of locking on the latency of a business activity are more significant. In the analysis below, we assume that a business activity is a sequence of tasks. We examine the effects of locking on the probability density function (pdf) of the latency of a business activity, based on a discrete delta function for the pdf for each task of the business activity, that is, effectively, we assume that each task has unit duration including both processing and communication times. Alternatively, we could employ a negative exponential pdf or a Gaussian pdf for each task of the business activity, with an increase in mathematical formulas, a decrease in clarity of exposition, and little improvement in the faithfulness to a particular business activity. To perform a precise analysis for a specific business activity, one would use a measured pdf for the duration of each task of that business activity, information that is not available to us.

We assume further that for classical transactional locking, in each task of the business activity, zero or more locks are claimed and are held until the end of the business activity. The probability that claiming a lock causes a lock conflict is a parameter of the analysis. If a lock is held longer because a business activity is blocked, waiting for another lock, the probability that another business activity attempts to claim that lock increases proportionately. Again, to perform a precise analysis for a particular business activity,

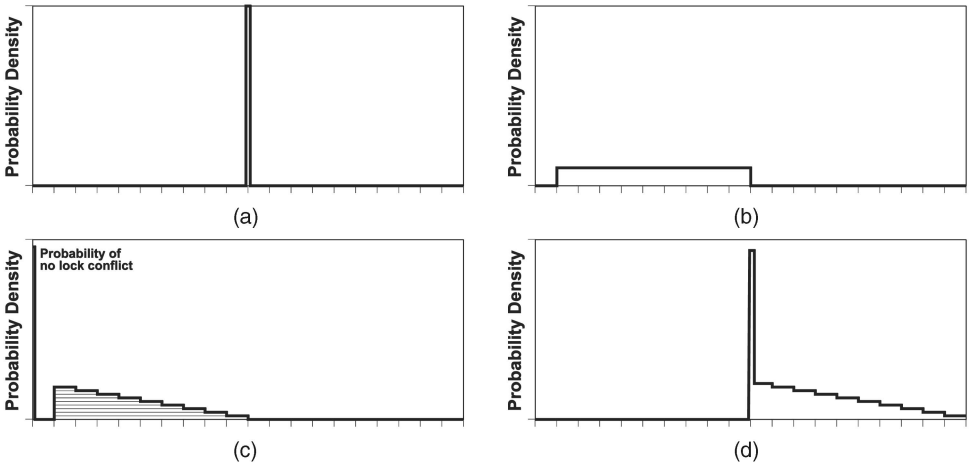

Fig. 6. The pdfs for the duration of a business activity, assuming that there are no delays due to lock contention, for the period for which a lock is held, for the delay resulting from lock contention, and for the duration of the business activity including delays resulting from lock contention. (a) Duration of Business Activity. (b) Duration for which a lock is held. (c) Duration of delay due to lock conflict. (d) Duration of Business Activity including delays.

one would use specific measurements for that business activity.

Fig. 6 shows the analysis required for determining the effect of lock contention on the duration of a business activity. Fig. 6a shows the pdf for the duration of a business activity assuming no delays due to lock contention. In this example, we assume that the duration of a business activity with no delays due to lock contention is 10 time units.

Fig. 6b shows the pdf for the period for which a lock is held, uniform between the minimal holding time and the entire duration of the business activity. Locks are claimed at random during the business activity and are held until the business activity completes.

Fig. 6c shows the pdf for the delay resulting from lock contention, assuming that claiming a lock results in lock contention. Each possible time for which the lock can be held, in Fig. 6b, contributes toward the delay shown in Fig. 6c. That contribution is uniform between the minimal delay and the time for which the lock is held. Note that as a lock is held longer, there is an increase not only in the potential delay but also in the probability of attempting to claim the lock, increasing the probability of incurring a delay.

The pdf for the delay due to lock contention, shown in Fig. 6c, is convolved with the pdf for the duration of the business activity without lock contention, shown in Fig. 6a, to produce the pdf for the business activity with lock contention, shown in Fig. 6d. Additional convolutions are required to represent the less likely possibility that the business activity is delayed by two or more locks.

However, the pdf for the period for which a lock is held and the pdf for the delay resulting from lock contention should be derived from the pdf for the duration of the business activity with lock contention, rather than from the pdf for the duration of the business activity without lock contention, which is the analysis we perform below.

Thus, we let $p(t)$ represent the pdf for the duration t of a business activity, $p_0(t)$ represent the pdf for the duration t of a business activity without lock contention, and q represent the probability of contention for a lock. Then, the pdf $h(t)$ for the period t for which a lock is held is given by

$$h(t) = \sum_{s=1}^{t} p(s),$$

and the pdf $d_1(t)$ for the delay t resulting from contention for a single lock is given by

$$d_1(t) = q \sum_{s=1}^{t} h(s) \text{ for } t > 0 \text{ and } d_1(0) = 1 - \sum_{s=1}^{\infty} d_1(s).$$

Similarly, the pdf $d_2(t)$ for the delay t resulting from contention for a second lock is given by

$$d_2(t) = d_1(t) \sum_{s=1}^{\infty} d_1(s) \text{ for } t > 0 \text{ and } d_2(0) = 1 - \sum_{s=1}^{\infty} d_2(s),$$

with corresponding formulas for $d_3(t)$, $d_3(0)$, etc.

Now, the pdf $p_1(t)$ for the duration t of a business activity with contention for at most one lock is derived as the convolution

$$p_1(t) = \sum_{s=1}^{t} p_0(s) d_1(t-s),$$

and the pdf $p_2(t)$ for the duration t of a business activity with contention for at most two locks is derived as the convolution

$$p_2(t) = \sum_{s=1}^{t} p_1(s) d_2(t-s),$$

with corresponding formulas for $p_3(t)$, etc.

Finally, the pdf $p(t)$ for the duration t of a business activity with contention for an arbitrary number of locks is given by

$$p(t) = p_\infty(t).$$

Fig. 7 illustrates the pdfs $p(t)$ for the duration t of a business activity for different values q of lock contention. Note that when the probability of lock contention is low, the pdfs for the duration of a business activity using classical transactional locking are substantially as expected, and the effects of delays due to contention for a single lock and for

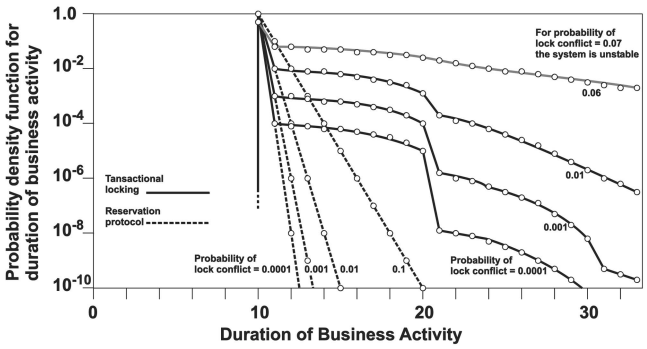

Fig. 7. The pdfs for the duration of a business activity with delays due to lock contention for both classical transactional locking and the reservation protocol.

two locks are clearly visible. As the probability of contention for a lock increases, the business activities are delayed, locks are held longer, delays due to lock contention are longer, and the probability that a business activity claims a lock that is already held by another business activity increases. The resulting pdfs have long tails, and thus, there is a high probability of lengthy delays for the business activity.

It is also worth noting that for transactional locking, there are probabilities for lock contention for which the system is not stable, representing unbounded delays and essentially no progress for the business activity. Such lock contention and instability leads to system collapse under a heavy load, which can occur at the most inappropriate times during the most important tasks. This observation underscores the importance of determining and enforcing an admission control limit for the business activities.

Also shown in Fig. 7 are the pdfs for the duration of a business activity for the reservation protocol. It is evident that even high probabilities of lock contention do not result in substantial delays for the business activity, because locks are held only briefly during the reservation subtask and are not held for the full duration of the business activity. In summary, the reservation protocol is more resilient to high loads and high probabilities of lock contention than is classical transactional locking.

The pdfs shown in Fig. 7 are intended to illustrate only the impact of locking on the latency of business activities in general and should not be applied to any particular business activity. For precise latency results, one must measure and analyze each business activity individually.

The locking of an available resource by a business activity that needs only some of the resources is undesirable, and thus, some businesses employ reservations within transactions. This mixed strategy is not easy to implement because in addition to implementing a reservation protocol similar to that described here, the mechanisms of transactional locking must be disabled for business activities that involve such reservations to allow other business activities to access the database records (resources) and to prevent the records from being restored to the previous state if a transaction is aborted.

If, instead of distributed transactions with transactional locking, only local transactions and compensating transactions are used, the reservation of only some of the resources is simpler. The concurrency achieved is substantially equivalent to that of the reservation protocol strategy.

7 Related Work

Extended transaction models [23], [30] are important today and will become more important in the future for business-to-business (B2B) systems. Such systems enable the automation of business processes across different enterprises. We have already provided detailed comparisons of our reservation protocol with other extended transaction strategies (including the escrow transactional method) in Section 4, and we do not repeat those comparisons here.

There are several standards for business process management, such as the Business Process Execution Language for Web Services (BPEL4WS), [2] and for business activity coordination, such as the WS-BA Framework [5]. The WS-BA specification has incorporated the sagas-based extended transaction protocol [7]. However, as we have already pointed out, compensation-based extended transaction protocols have their limitations and problems. We have presented an alternative reservation-based protocol to overcome those limitations and problems. In addition to the standards, there exists commercial middleware for workflow management, such as the WebSphere MQ Workflow Middleware [15] and the BizTalk Server [21].

Closely related to our reservation protocol is the OASIS Business Transaction Protocol (BTP) [24], in particular, the BTP cohesion protocol. The BTP cohesion protocol is a two-phase protocol in which the business transaction participants have explicit control over the two phases. In the first phase, all of the participants are required to prepare, that is, they must ensure that a task can be committed or rolled back if a fault occurs. In the second phase, the coordinator issues confirmation or cancellation requests to the participants. One might argue that our reservation subtask is a special form of the prepare phase and, thus, that our protocol is a special implementation of the BTP cohesion protocol. However, the BTP specification mentions a reservation-like action as only one of several possible ways to provide provisional or tentative state changes. The BTP specification does not pursue the concept of reservation to the same extent that we do and does not elaborate the benefits of the reservation approach.

Another closely related work is the atomic reservation protocol for reserving resources in a free market, described by Ginis and Chandy [10]. In their protocol, a consumer makes timed-reservation requests to the service providers in the form of purchasing options for the use of resources that the service providers hold. In the second phase, confirmation/cancellation requests are sent to the service providers. Compensating transactions are used to negate the effects of a partially fulfilled plan and to cope with faults and the expiration of options acquired during the first phase. The start of the first phase and the completion of the second phase are assumed to be short in duration. Their protocol is not necessarily appropriate for loosely-coupled long-running business activities for which our reservation protocol is specifically designed.

The tentative holding protocol (THP) [25], [26] is used to exchange information between enterprises before a transaction begins. THP allows tentative nonblocking holds

(reservations to be requested) for a business resource. It minimizes the possibility of rolling back committed transactions by providing more accurate information regarding the availability of the resource to the client. Unlike our reservation protocol, THP allows multiple clients to hold the same resource temporarily. When one of the clients places an order, the remaining clients receive notifications of the unavailability of the resource. However, nothing prevents a client from placing an order for a resource immediately, at which point another client might have already taken the resource. In such a case, the business activity must be rolled back, and the client must apply a compensating transaction to cancel the previously committed transaction. In our reservation protocol, the reservation subtask is part of the business activity, the reservation is granted exclusively to a single client, and blocking reservations avoid the need for compensating transactions.

B2B systems often demand higher availability and performance than can be provided by any particular extended transaction protocol, and additional mechanisms might have to be built into the middleware and the applications. Wachter and Reuter have proposed the ConTract model [27] for defining and controlling complex long-lived activities at a level above ACID transactions. Unlike conventional programming languages, the ConTract model is a programming model that includes persistency, consistency, recovery, synchronization, cooperation, and the use of assertions as invariants on entry to/exit from activities.

Greenfield et al. have been engaged in a project on maintaining consistency in loosely-coupled distributed environments. In [13], they discuss characteristics of B2B applications, existing mechanisms and their inadequacies, and consistency requirements for B2B applications. In [14], they provide a set of requirements for a cancellation mechanism for an e-procurement case study. In [6], they propose a research agenda for the development of an infrastructure that includes a language to express consistency conditions, tools to check whether the system maintains consistency, and guidance for using the infrastructure properly.

Other researchers have investigated the performance of enterprise computing systems. Gillmann et al. [8], [9] provide a transient analysis of a system's performance for different degrees of replication of different types of servers (workflow engines, application servers, and communication servers) using continuous-time Markov models. Our analysis differs from their analysis in that we compare the effects of using different kinds of architectures on the performance of a distributed application, rather than the effects of different degrees of replication of the servers. In [18], Klingemann et al. provide a steady-state analysis based on continuous-time Markov models for workflow management to assess the efficiency of different outsourcing strategies in a virtual enterprise setting. Our analysis is quite different from their analysis in its objective. Moreover, the use of continuous-time Markov models is appropriate in a context where measurements of continuous-time pdfs for the durations of the tasks of business activities are available. We lack that information and consequently have used a discrete-time analysis for the latency of the business activities.

8 Conclusions and Future Work

Business activities need extended transaction protocols to coordinate tasks in order to achieve coherent results. In the current state of the art, compensating transactions are used as a means of handling faults and exceptions when they occur. In this paper, we have described a novel reservation-based extended transaction protocol that can be used to coordinate the tasks of long-running business activities. Instead of resorting to compensating transactions, our reservation protocol employs an explicit reservation subtask for each task that is not read-only to allow for the possibility of later changes. Thus, each task is executed as two subtasks. The first subtask involves an exclusive blocking reservation of the resource. The second subtask involves the confirmation or cancellation of the reservation. As our analyses show, the reservation protocol exhibits performance similar to or better than that of other strategies. Future work includes the design and implementation of a software infrastructure at the middleware layer that integrates our reservation protocol with commercial off-the-shelf workflow engines such as [15] and [21] in order to enhance the performance of business activities in a loosely-coupled distributed environment.

References

[1] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Gunthor, and C. Mohan, "Advanced Transaction Models in Workflow Contexts," *Proc. 12th Int'l Conf. Data Eng. (ICDE '96),* pp. 574-581, Feb.-Mar. 1996.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, *Business Process Execution Language for Web Services Version 1.1,* http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, 2007.

[3] R. Barga, D. Lomet, G. Shegalov, and G. Weikum, "Recovery Guarantees for Internet Applications," *ACM Trans. Internet Technology,* vol. 4, no. 3, pp. 289-328, 2004.

[4] L.F. Cabrera, G. Copeland, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, D. Langworthy, A. Nadalin, D. Orchard, I. Robinson, T. Storey, and S. Thatte, *Web Services Atomic Trans.,* http://www.ibm.com/developerworks/library/ws-transpec/, 2007.

[5] L.F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, F. Leymann, I. Robinson, T. Storey, and T. Thatte, *Web Services Business Activity Framework,* http://www.ibm.com/developerworks/library/ws-busact/, 2007.

[6] A. Fekete, P. Greenfield, D. Kuo, and J. Jang, "Transactions in Loosely Coupled Distributed Systems," *Proc. 14th Australasian Database Conf. (ADC '03),* pp. 7-12, Feb. 2003.

[7] H. Garcia-Molina and K. Salem, "Sagas," *Proc. ACM SIGMOD '87,* pp. 249-259, 1987.

[8] M. Gillmann, J. Weissenfels, G. Weikum, and A. Kraiss, "Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems," *Proc. Seventh Int'l Conf. Extending Database Technology (EDBT '00),* pp. 183-201, Mar. 2000.

[9] M. Gillmann, G. Weikum, and W. Wonner, "Workflow Management with Service Quality Guarantees," *Proc. ACM SIGMOD '02,* pp. 228-239, June 2002.

[10] R. Ginis and K.M. Chandy, "Micro-Option: A Method for Optimal Selection and Atomic Reservation of Distributed Resources in a Free Market Environment," *Proc. Second ACM Conf. Electronic Commerce (EC '00),* pp. 207-214, 2000.

[11] C. Hagen and G. Alonso, "Exception Handling in Workflow Management Systems," *IEEE Trans. Software Eng.*, vol. 26, no. 10, pp. 943-958, 2000.
[12] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.
[13] P. Greenfield, A. Fekete, J. Jang, and D. Kuo, "What Are the Consistency Requirements for B2B Systems," *Proc. High Performance Transaction Systems Workshop (HPTS '03),* Oct. 2003.
[14] P. Greenfield, A. Fekete, J. Jang, and D. Kuo, "Compensation Is Not Enough," *Proc. Seventh IEEE Int'l Enterprise Distributed Object Computing Conf. (EDOC '03),* pp. 232-239, Sept. 2003.
[15] IBM, *WebSphere MQ Workflow Middleware,* http://www.ibm.com/software/integration/wmqwf/, 2007.
[16] IBM, BEA, Microsoft, and Tibco, *Web Services Reliable Messaging (WS-RM),* http://www-128.ibm.com/developerworks/webservices/library/ws-rm/, 2007.
[17] M. Kamath and K. Ramamritham, "Failure Handling and Coordinated Execution of Concurrent Workflows," *Proc. 14th Int'l Conf. Data Eng. (ICDE '98),* pp. 334-341, Feb. 1998.
[18] J. Klingemann, J. Waesch, and K. Aberer, "Deriving Service Models in Cross-Organizational Workflows," *Proc. Ninth Int'l Workshop Research Issues on Data Engineering (RIDE '99)—Information Technology for Virtual Enterprises,* pp. 100-107, Mar. 1999.
[19] N. Lynch, "Multilevel Atomicity—A New Correctness Criterion for Database Concurrency Control," *ACM Trans. Database Systems,* vol. 8, no. 4, pp. 484-502, 1983.
[20] C.P. Martin and K. Ramaritham, "Guaranteeing Recoverability in Electronic Commerce," *Proc. Third Int'l Workshop Advanced Issues of E-Commerce and Web-Based Information Systems,* pp. 144-155, June 2001.
[21] Microsoft, *BizTalk Server,* http://www.microsoft.com/biztalk/, 2007.
[22] J.E.B. Moss, *Nested Transactions: An Approach to Reliable Distributed Computing.* MIT Press, 1985.
[23] P.E. O'Neil, "The Escrow Transactional Method," *ACM Trans. Database Systems,* vol. 11, no. 4, pp. 405-430, 1986.
[24] Organization for the Advancement of Structured Information Standards (OASIS), *Business Transaction Protocol, Version 1.0,* http://www.oasis-open.org/committees/businesstransactions/, 2007.
[25] J. Roberts, T. Collier, P. Malu, and K. Srinivasan, *Tentative Hold Protocol Part 2: Technical Specification,* http://www.w3.org/TR/tenthold-2, 2007.
[26] J. Roberts and K. Srinivasan, *Tentative Hold Protocol Part 1: White Paper,* http://www.w3.org/TR/tenthold-1, 2007.
[27] H. Wachter and A. Reuter, "The ConTract Model," *Database Transaction Models for Advanced Applications,* A.K. Elmagarmid, ed., Morgan Kaufmann, pp. 219-263, 1992.
[28] G. Weikum and H. Schek, "Concepts and Applications of Multilevel Transactions and Open Nested Transactions," *Database Transaction Models for Advanced Applications,* A. Elmagarmid, ed., Morgan Kaufmann, pp. 515-553, 1992.
[29] G. Weikum, "Principles and Realization Strategies of Multilevel Transaction Management," *ACM Trans. Database Systems,* vol. 16, no. 1, pp. 132-180, 1991.
[30] G. Weikum, "Extending Transaction Management to Capture More Consistency with Better Performance," *Proc. Ninth French Database Conf.,* pp. 251-276, Sept. 1993.
[31] D. Worah and A. Sheth, "Transactions in Transactional Workflows," *Advanced Transaction Models and Architectures,* S. Jajodia and L. Kershberg, eds., pp. 3-34, 1997.
[32] W. Zhao, L.E. Moser, and P.M. Melliar-Smith, "Unification of Transactions and Replication in Three-Tier Architectures Based on CORBA," *IEEE Trans. Dependable and Secure Computing,* vol. 2, no. 1, pp. 20-33, 2005.
[33] W. Zhao, L.E. Moser, and P.M. Melliar-Smith, "A Reservation-Based Coordination Protocol for Web Services," *Proc. IEEE Int'l Conf. Web Services (ICWS '05),* pp. 49-56, July 2005.