Maintaining Consistency of Replicated Data in Multidatabase Systems

Jin Jing*, Weimin Du†, Ahmed Elmagarmid‡, and Omran Bukhres‡
* Intergraph Corporation, Huntsville, AL 35894
† Hewlett-Packard Labs., Palo Alto, CA 94304
‡ Purdue University, West Lafayette, IN 47907

Abstract

The paper presents two protocols for maintaining consistency of replicated data in multidatabase systems. The protocols meet both autonomy and consistency requirements by employing different replica control and commitment approaches: unilateral local commitment and deferred propagation for local applications, and two-phase commitment and immediate propagation for global applications. The first protocol ensures replication consistency (with regard to one copy serializability) through the use of a global certification protocol. The second protocol, which is an extension of the first by using propagation locks on primary copies, allows consistency certification to be performed locally for global queries that only read data copies from a single site. We identify and examine the major issues relevant to the proposed protocols.

1 Introduction

A multidatabase system (MDBS) is a federation of pre-existing database systems (called local database systems, or LDBSs), which includes both *global* and *local* applications. Local applications were developed on a single LDBS before the construction of the MDBS, while global applications are developed afterwards and usually access data from multiple LDBSs. One of the important features of an MDBS is the autonomy of its LDBSs. Local autonomy is both desirable and necessary in MDBSs; it facilitates the flexible interconnection of various kinds of LDBSs and guarantees that local applications are executable after interconnection.

The main objective of interconnecting otherwise isolated LDBSs is the sharing of data. An effective approach to data sharing among LDBSs is the replication of local data at remote sites. Replication allows an application to read otherwise "remote" data efficiently, thereby reducing retrieval costs and increasing the data availability.

However, the maintenance of local autonomy causes transaction processing in replicated MDBSs to be more difficult than in traditional distributed databases. For example, it may not be appropriate to execute and commit a local application within a single transaction that both updates and propagates replicated data[DEKB93]. The execution of such a transaction depends upon an atomic commitment protocol, such as two-phase commit protocol, which is under the control of a global coordinator. Although this difficulty can be circumvented by propagating updates asynchronously (by a separated transaction), this complicates other issues of transaction processing, such as the maintenance of replication consistency and the optimization of performance for global applications. For instance, it may not be possible to guarantee that the nearby or local copies of replicated data have been made consistent with a global application prior to the completion of deferred propagating updates.

The primary goal of the paper is to investigate the issue and propose an effective approach to replicated data management that preserves local autonomy, maintains replication consistency, and optimizes transaction performance. In the proposed approach, local applications (regardless of replication) should be executable independently, without consulting the global system and other LDBSs; Global applications should also be free to access any copy of replicated data without compromising one-copy serializability (1SR) [BHG87]; Furthermore, single-site read-only global applications should be executed locally, without the control of the global transaction manager.

We present two protocols for replicated data management which employ deferred propagation for local applications and immediate propagation for global applications. The first protocol ensures replication consistency (with regard to 1SR) through the use of a global certification protocol which verifies the consistency of replicated copies accessed by both local and global applications. The second protocol, an extension of the first, uses propagation locks on primary copy sites. The propagation locks permit the consistency certification to be performed locally for single-site query applications, thereby improving the performance of transaction processing.

The relevance of the work presented here is highlighted by the increase of interest in multidatabase systems and by the lack of practical methods which maintain the consistency of replicated copies, optimize transaction performance, and preserve autonomy. Although autonomy has always been recognized as an important issue in traditional distributed database systems, most of proposed methods have fo-

1063-6927/94 $03.00 © 1994 IEEE

cused on other issues, such as replication consistency and performance optimization and compromise autonomy. These methods usually update and propagates replicated data within a single transaction, using a two-phase-commit protocol[CL91].

The protocol described in [Ston79] allows the immediate update of primary copies and performs the propagation of updates asynchronously in a separated transaction. This protocol supports a high degree of autonomy for the management of replicated data. However, the method is applicable only for those applications where weaker consistency requirements (than 1SR) are acceptable.

[WQ90] presented a conceptual framework for the control of the consistency of replicated data in MDBSs. This framework specifies a variety of consistency requirements and employs a asynchronous *promise protocol* to enforce consistency requirements in non-primary copy sites. The proposal, however, did not demonstrate how to implement the consistency enforcement which ensures 1SR of transactions executed in an MDBS environment. Moreover, we feel that the promise protocol itself may compromise local autonomy, because the primary copy site must wait for the acknowledgement (the promise) from non-primary copy sites before it can commit.

This work is most closely related to that presented in [DEKB93]. The two works share the same replicated MDBS model, as well as many other basic techniques such as immediate update of the primary copy and deferred propagation to non-primary copies for *local* applications. The major difference is the consistent access to non-primary copies by *global* applications. The protocol presented in [DEKB93] limited access of global transactions to primary copies to avoid inconsistent execution. Such a limitation increases global query costs and restricts the availability of replicated data, as the query may have to access remote primary copies rather than replicated local copies. By contrast, the proposed work allows global applications to access any local or nearby non-primary copy. The motivation for the work is firmly consistent with the objectives of data replication: to improve the availability of replicated data and to increase the performance of transaction processing.

The rest of the paper is organized as follows. Section 2 presents the system model for replicated MDBSs to be employed in this paper. The consistency certification protocol is presented in Section 3. Section 4 addresses the issue of globally uncontrolled single-site queries and illustrates a propagation lock approach to ensure 1SR for these queries. Finally, Section 5 provides brief concluding remarks.

2 The System Model

A replicated MDBS consists of a set of *local database systems* $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ and a set of transactions $\mathcal{T}$. Each local database system is defined by a pair of data sets $S_i =< \mathcal{D}_i, \mathcal{P}_i >$, where $\mathcal{D}_i$ is a set of *logic data items* belonging to S_i (which includes all data originally residing at S_i before they were replicated), while $\mathcal{P}_i$ is a set of *physical data copies* residing at S_i after the replication. We also use $\mathcal{D} = \cup_{i=1}^{n}\mathcal{D}_i$ and $\mathcal{P} = \cup_{i=1}^{n}\mathcal{P}_i$ to denote, respectively, the set of logical data items that belong to and the set of physical copies that reside at the MDBS. The function $f : \mathcal{D} \rightarrow 2^{\mathcal{P}}$ (where $2^{\mathcal{P}}$ denotes the powerset of $\mathcal{P}$) gives, for each data item d, the set of data copies which implement d. A data item d is said to be replicated if the cardinality of $f(d)$, denoted $\mid f(d) \mid$, is greater than one (i.e., $\mid f(d) \mid > 1$). Otherwise, it is said to be non-replicated (i.e., $\mid f(d) \mid = 1$). We use d_i^j to denote the physical copy of data item d_i at site S_j. For a replicated data item $d_i \in \mathcal{D}_i$, there always exists a data copy $d_i^i \in \mathcal{P}_i \cap f(d_i)$ (i.e., the copy that resides at its host site) which is called the *primary copy*.

As mentioned previously, there are two kinds of applications in a replicated MDBS: *global* and *local* applications. Local applications were developed on a single LDBS before the construction of the MDBS and therefore access only a single $\mathcal{D}_i$, while global applications are developed afterwards and usually access multiple $\mathcal{D}_i$s. Global applications may need to be implemented as *global* transactions (or multisite transactions), while local applications are implemented either as a *basic local* transaction which does not update replicated data, or as a *replica-update local* transaction, which updates both replicated and non-replicated data. From a user's point of view, replica-update and basic local transactions present the same appearance, as they access only data items that logically belong to a single LDBS. From a transaction management view point, however, replica-update transactions appear to be global, as they access data copies that physically reside at several sites. In the rest of this paper, these two situations will simply be referred to as local and replica-update transactions.

Formally, the transaction set $\mathcal{T}$ consists of $\mathcal{GT}$, $\mathcal{LT}_1$, $\mathcal{RT}_1$, $\mathcal{LT}_2$, $\mathcal{RT}_2$, ..., $\mathcal{LT}_n$, and $\mathcal{RT}_n$, where $\mathcal{GT}$ is the set of global transactions, $\mathcal{LT}_i$ is the set of local transactions belonging to S_i, and $\mathcal{RT}_i$ is the set of replica-update transactions belonging to S_i. A global transaction G_i consists of a set of subtransactions (called *global subtransactions*) $G_{i,1}, G_{i,2}, ..., G_{i,n}$, where $G_{i,j}$ accesses copy $d_i^j \in \mathcal{P}_j$ only (We use $T_{i,j}$ to denote the subtransaction of T_i at site S_j). A local transaction accesses copy $d_i^i \in \mathcal{P}_i \cap f(\mathcal{D}_i)$ only where $\mid f(d_i^i) \mid = 1$. Similarly, a replica-update transaction R_i consists of a *primary-update* subtransaction $R_{i,i}^U$ which accesses copy $d_i^i \in \mathcal{P}_i \cap f(\mathcal{D}_i)$ only where $\mid f(d_i^i) \mid \geq 1$, and a set of *propagation-update* subtransactions $R_{i,1}^P$, $R_{i,2}^P$, ..., $R_{i,n}^P$, where $R_{i,j}^P$ accesses copy $d_i^j \in f(\mathcal{D}_i) \cap \mathcal{P}_j$ $(i \neq j)$ only where $\mid f(d_i^j) \mid > 1$. We use the term propagation transaction, denoted R_i^P, to refer to the set of propagation-update subtransactions of a replica-update transaction R_i.

We use $\mathcal{RT} = \cup_{i=1}^{n}\mathcal{RT}_i$ and $\mathcal{LT} = \cup_{i=1}^{n}\mathcal{LT}_i$ to denote, respectively, the set of all replica-update transactions and the set of all local transactions in the MDBS. We also use $\mathcal{RU} = \{R_{i,i}^U \mid R_i \in \mathcal{RT}\}$ to denote the set of all primary-update subtransactions, and $\mathcal{RP} = \{R_i^P \mid R_i \in \mathcal{RT}, i \neq j\}$ to denote the set of all propagation transactions.

An important assumption in this model is that logical data items in two sites are disjoint, i.e., $\mathcal{D}_i \cap \mathcal{D}_j = 0$ where $(i \neq j)$ and a local application accesses a single $\mathcal{D}_i$ only. The assumption reflects the fact that local applications at different sites were independently developed prior to the creation of the MDBS and that they can still be executed independently, with no involvement by the global system, if no global application is involved. For example, in such a case, two primary-update subtransactions at different sites can be executed and committed locally without the control of the global system. Furthermore, their propagation-update subtransactions also can be executed independently, with consistency preserved (in terms of 1SR).

The transaction processing model used in this paper is presented in Figure 1. There are three major components: a global data manager (GDM), a global transaction manager (GTM), and a set of servers residing at each local site.

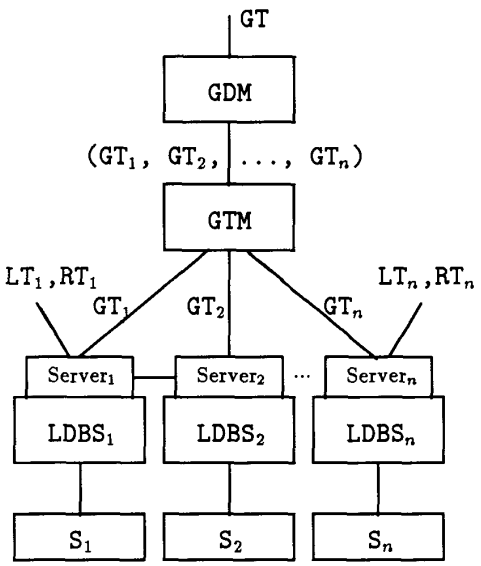

Figure 1: Transaction Processing in Replicated MDBSs

Both the GDM and the GTM are centrally located, while a server is superimposed on the LDBS at each site. The interrelationship of these components is as follows. A global application (in logical operations) is first submitted as a global transaction to the GDM, which decomposes it into a set of subtransactions (in physical operations) and passes them to the GTM. The GTM then sends the subtransactions to servers at local sites and coordinates their execution. Each server in turn sends each operation (including commit/abort) of the subtransaction to the underlying database, as instructed by the GTM.

In this model, local applications must first be submitted to the server, rather than directly to the underlying database. The server translates each logical operation on a data item into the corresponding physical operation on the local primary copies of the data. If the application updates replicated data, it is executed as a replica-update transaction. The server submits its primary-update subtransaction to the local DBMS and then forms a set of propagation-update subtransactions which are forwarded to the GTM. The GTM executes and monitors these subtransactions like a global transaction. If the local application does not update any replicated data, then it is submitted to the local DBMS as a local transaction. One of advantages of using servers to translate logical data into physical copies and to form propagation transactions is that there is no need to modify local application codes after the construction of the MDBS.

3 A Certification Protocol

3.1 Motivation

In order to preserve local autonomy, LDBSs schedule primary-update subtransactions independently and commit them unilaterally, regardless of update propagation. The unilateral commitment and deferred propagation, however, presents a challenge to global concurrency control which ensures 1SR. The challenge becomes manifest when global transactions, for reasons of availability and performance, access nearby non-primary copies. These copies may be latently inconsistent with their primary copies due to the deferred propagation.

Example 3.1 *Consider a replicated MDBS where S_1 has a primary copy d^1 of data item d and S_2 has non-primary copy d^2 of d. Suppose that both global transaction G_1 ($G_{1,1} : w_{g_1}(d^1)$ and $G_{1,2} : r_{g_1}(d^2) w_{g_1}(d^2)$) and replica-update transaction R_1 ($R^U_{1,1} : r_{r_1}(d^1) w_{r_1}(d^1)$ and $R^P_{1,2} : w_{r_1}(d^2)$) read and write the item d. The following execution is not one-copy serializable.*

$E_1 : r_{r_1}(d^1) w_{r_1}(d^1) w_{g_1}(d^1) p_{g_1} c_{r_1} c_{g_1}$

$E_2 : r_{g_1}(d^2) w_{g_1}(d^2) w_{r_1}(d^2) p_{g_1} c_{g_1} c_{r_1}$.

The example illustrates that if 1SR is to be ensured, R_1 must be treated like a global transaction. A coordination protocol must ensure that R_1 and G_1 have a consistent serialization order in both sites.

To the best of our knowledge, no currently-proposed global protocol satisfactorily handles such a situation in which primary-update subtransactions are committed unilaterally. Some protocols require the GTM to dynamically make commit decisions for all global transactions (e.g., optimistic approaches), while others statically assign transactions a serialization order which local servers must enforce (e.g., pessimistic approaches). The approaches do not work well in replicated MDBSs, as the GTM could neither make a commit decision for a primary-update subtransaction nor assign it a serialization order before it is submitted to local sites. All primary-update subtransactions are executed independently without the coordination of GTM.

In the subsection 3.2, we will present a two-phase certification protocol addressing the problem. In this

protocol, the GTM certifies the existence of $R_{1,1}$ before it can commit G_1. The certification is performed during the first phase of the two phase commit protocol, during which G_1 enters into its prepared state at both S_1 and S_2. This knowledge of the existence of $R_{1,1}$ is essential for the GTM to ensure that R_1 is serialized before (or after) G_1 at both sites. The certification protocol also guarantees that any two global or propagation transactions will have a consistent serialization order in all sites. Subsection 3.3 discusses the basic requirements for local concurrency control in our proposed protocol.

3.2 Two-Phase Certification Protocol

The following concepts will be useful in describing the certification protocol.

A *global total order* O_g is a linear order over all global and replica-update transactions. A *local serialization order* O_l^k at Site S_k is a serialization order over global and primary-update/propagation-update subtransactions executed at the site. We say that two orders O and O' are *compatible* if, for any two transactions T_1 and T_2 that appear in both orders, either T_1 precedes T_2 or T_2 precedes T_1 in both O and O'.

A global transaction G_i is *globally committed* if all its subtransactions have committed. A replica-update transaction R_i is *globally committed* if both $R_{i,i}^U$ and R_i^P have committed.

Clearly, an execution of global and replica-update transactions is *one-copy serializable* if these executions are globally committed and there exists a global total order O_g which is compatible with local serialization order O_l^k from site S_k for all k. We assume that each LDBS guarantees local serializability. Therefore, There always exists such a local serialization order O_l^k at S_k. We further assume that each LDBS provides a visible *prepared-to-commit* state for its transactions. The assumption implies that an LDBS will not unilaterally abort a subtransaction after it has completed all its operations and enters into the prepared-to-commit state.

In the proposed protocol, a global transaction is submitted to the GTM where it is decomposed and sent to local servers. A primary-update subtransaction is submitted locally to the server. After the primary-update subtransaction finishes its execution, it is immediately committed at the local DBMS. A propagation transaction is then formed and sent to the GTM. The GTM monitors and executes the propagation transaction just like a regular global transaction.

The server is responsible for the execution coordination of subtransactions (global, primary-update, and propagation-update) and for the determination of their serialization order O_l^k at S_k. The GTM collects local serialization orders from local servers and certifies the compatibility of these orders before a global or a propagation transaction can be committed.

The collection and certification of serialization orders are performed concurrently through the use of the 2PC protocol. During the first phase of the 2PC protocol, the server at S_k sends to the GTM a READY message and a serialization order O_l^k, which includes all locally-committed primary-update subtransactions and prepared global or propagation-update subtransactions at the site. In the second phase, the GTM checks the compatibility of these orders and the atomicity of replica-update transactions before they are committed.

More specifically, after receiving READY messages for all subtransactions of a global transaction G_i or a propagation transaction R_i^P, the GTM sends COMMIT messages to servers only if the following two conditions are true; otherwise, the GTM sends ABORT messages to abort the transaction.

Condition 3.1 *There exists a global total order O_g which is compatible with local serialization order O_l^k for all k.*

Condition 3.2 *All replica-update transactions preceding G_i or R_i^P in O_g have been globally committed.*

Condition 3.1 implies that those prepared or committed global or replica-update transactions in O_g are globally serializable, while Condition 3.2 guarantees that a primary-update subtransaction and its corresponding propagation-update subtransactions can be globally committed as a single transaction. In order for GTM to properly check both Condition 3.1 and Condition 3.2 during the first phase, the server at S_k must satisfy the following two properties:

Property 3.1 *The server can derive the serialization order of* prepared *subtransactions (global, primary-update, or propagation-update) in the local DBMS.*

Property 3.2 *The server should schedule two subtransactions (global, primary-update, or propagation-update) in such a manner that their serialization order is consistent with the order in which they enter into the PREPARED state, if one (and only one) of them is a primary-update subtransaction.*

The need for Property 3.1 is obvious, as servers must send the serialization orders O_l^k for all k of prepared subtransactions to the GTM during the first phase of 2PC protocol. We will elaborate upon the process of deriving these orders at server level in Section 3.3.

Property 3.2 is necessitated by the fact that primary-update subtransactions are not directly controlled by the GTM. The property ensures that the GTM has sufficient information about the atomicity of replica-update transactions preceding G_i at the time when G_i is to be committed.

The second property is more restrictive than the first and can be guaranteed only if servers can derive local serialization orders when subtransactions enter into a PREPARED state. The two properties are formulated separately because they serve different purposes in the certification protocol; the first property is involved in the construction of local serialization orders, while the second ensures the atomicity of replica-update transactions. The violation of the second property may result in a non-atomic execution, as the following example shows.

Example 3.2 *Consider the MDBS in Example 3.1. Suppose the server at S_1 executes $G_{1,1}$ and $R^U_{1,1}$ concurrently and $R^U_{1,1}$ precedes $G_{1,1}$ in the local serialization order. If $R^U_{1,1}$ enters the prepared state after $G_{1,1}$, then the following non-atomic execution is possible:*

$E_1 : r_{r_1}(d^1)w_{r_1}(d^1)w_{g_1}(d^1)p_{g_1}p_{r_1}c_{r_1}c_{g_1}$

$E_2 : r_{g_1}(d^2)r_{g_1}(d^2)w_{g_1}(d^2)p_{g_1}c_{g_1}$.

In the execution, $R^U_{1,1}$ precedes $G_{1,1}$ in the serialization order in E_1, while $R^P_{1,2}$ does not appear before $G_{1,2}$ in E_2. Because $G_{1,1}$ entered the prepared state before $R^U_{1,1}$, the server in S_1 could not include $R^U_{1,1}$ in O^1_l which is attached to the READY message of $G_{1,1}$. As a consequence, $R^U_{1,1}$ may be committed before $G_{1,1}$ arrives the server, resulting a non-atomic execution of R_1.

Please note that for a primary-update subtransaction and a propagation-update subtransaction, it may be either unnecessary or impossible to enforce a serialization order which is consistent with their prepared order. In fact, because $R^U_{k,k}$ accesses only primary copies or non-replicated local data and $R^P_{j,k}$ does not access such data at S_k ($j \neq k$), the subtransactions do not directly conflict. Furthermore, local transactions will not cause them to indirectly conflict, as local transactions do not access any non-primary copy that $R^P_{j,k}$ updates. As a consequence, if there is no other global subtransaction $G_{i,k}$ serialized between the two subtransactions, they can be considered in any serialization order by both the server and the GTM, regardless of their prepared order. The example below illustrates such a case.

Example 3.3 *Consider a replicated MDBS where S_1 has a primary copy d^1_1 and a non-primary copy d^1_2 and S_2 has a primary copy d^2_2 and a non-primary copy d^2_1. Suppose that R_1 ($r_{r_1}(d_1)w_{r_1}(d_1)$) is decomposed by the primary-copy protocol into $R^U_{1,1}$ ($r_{r_1}(d^1_1)w_{r_1}(d^1_1)$) and $R^P_{1,2}$ ($w_{r_1}(d^2_1)$) and R_2 ($r_{r_2}(d_2)w_{r_2}(d_2)$) is decomposed into $R^U_{2,1}$ ($w_{r_2}(d^1_2)$) and $R^U_{2,2}$ ($r_{r_2}(d^2_2)w_{r_2}(d^2_2)$).*

Clearly, the execution orders of R_1 and R_2 at local sites have no impact on the global serializability, as they do not conflict each other and local transactions would not introduce any indirect conflict either.

$E_1 : r_{r_1}(d^1_1)w_{r_1}(d^1_1)w_{r_2}(d^1_2)p_{r_1}c_{r_1}p_{r_2}c_{r_2}$

$E_2 : r_{r_2}(d^2_2)w_{r_2}(d^2_2)w_{r_1}(d^2_1)p_{r_2}c_{r_2}p_{r_1}c_{r_1}$.

3.3 Deriving Serialization Order

In an autonomous MDBS environment, it is assumed that the data structures and functions involved with transaction management in local DBMSs are not available to the global system (e.g., the GTM). The maintenance of global serializability in an MDBS, therefore, depends fundamentally upon the ability to derive the local serialization order of subtransactions from the interface supported by local DBMSs. This remains a significant issue regardless of whether an optimistic or pessimistic global concurrency control protocol is used.

This issue has been the subject of extensive investigation in recent years [GRS91, BGS92, RAZ92]. The rigorous schedule method and the strongly recoverable schedule method [BGS92, RAZ92] derive the serialization order of subtransactions through their commitment order. These methods assume that local DBMSs use either rigorous schedule concurrency control or strongly recoverable concurrency control. The ticket method proposed in [GRS91] determines the serialization order at the global level using a special data item, termed a ticket, at each local site. Only global subtransactions are allowed to access the ticket at each site. The order of ticket access and the local conflicts arising from ticket access, therefore, determines a unique serialization order of subtransactions.

All these proposed methods, however, determines local serialization order on *commit events*. For example, the ticket method certifies local serializable order when global transactions are about to commit. Therefore, these methods may not be applicable to our two-phase certification protocol, which requires that servers derive the serialization order of subtransactions at time when they reached their PREPARED state. In the rest of this subsection, we will examine variations on these proposed methods on the basis of *prepare events*. The examination will include the feasibility of deriving the serialization order of subtransactions at the point of the prepared state, when a local rigorous schedule or a local strongly recoverable schedule will be assumed, or when local conflicts will be enforced through the use of tickets.

As a basis for this examination, we observe that if a local scheduler ensures serializability, a subtransaction enters into the prepared state only if it does not conflict inconsistently with other committed or prepared transaction. Otherwise, local serializability will not be ensured as all prepared transaction could commit and the local scheduler can not abort them unilaterally.

We further observe that a serialization order can be derived at prepare time for both rigorous schedulers and ticket based schedulers. However, it cannot always be derived for strongly recoverable schedulers.

In fact, the order can be easily derived at prepare time for ticket method, as the serialization order of global transactions is determined by ticket values read from the LDBSs. Clearly, a global subtransaction must have read the ticket value before it enters into PREPARED state.

When a local rigorous schedule is employed, the serialization order of global transactions is identical with the order in which they enter into the PREPARED state. As an illustration, consider two transactions T_i and, T_j, where T_i follows T_j in the serialization order. There exist transactions $T_1, T_2, ...,$ and T_l ($0 \leq l$) such that $T_j \prec T_1 \prec T_2 \prec ... \prec T_l \prec T_i$ ($T_p \prec T_q$ denotes that T_p directly conflicts with and precedes T_q in the serialization order). Let o_j, o_1, o_2, ..., o_l, and o_i be the conflicting operations of T_j, T_1, T_2, ..., T_l, and T_i, respectively.

By the definition of rigorous schedule protocol

[BGRS91], we have $o_j < c_j < o_1 < c_1 < ... < o_i < c_i$ ($a < b$ denotes that a is executed before b). Since $o_k < p_k < c_k$, we have $o_j < p_j < c_j < o_1 < p_1 < c_1 < ... < o_i < p_i < c_i$. So, $p_j < p_i$. That is, the order in which T_i and, T_j enter into the PREPARED state is identical with their serialization order.

The following example, however, shows that strongly recoverable schedulers are not always able to determine the serialization order of prepared subtransactions.

Example 3.4 *Consider the execution order O: $o_j < o_i < p_i < p_j < c_j < c_i$, where o_i and o_j are the two conflicting operations of transactions T_i and T_j, respectively.*

Since $o_j < o_i$ and $c_j < c_i$, by the definition, O is a serializable and strongly recoverable execution. However, $p_i < p_j$. The difference makes it impossible for the server to derive the local serialization order from the prepare order.

4 Processing Single-Site Queries

4.1 Problems

The advantage of the certification protocol presented in the previous section is that it maintains replication consistency without violating local autonomy. The protocol, however, requires that an application which accesses copies that logically belong to different sites be processed as a global transaction. As a consequence, the two-phase certification protocol must be used to control commitment, involving two complete message exchanges between the GTM and servers. Even a *single-site query* which only reads non-primary copies must be treated as a global transaction. Otherwise, the global 1SR will not be guaranteed, as the following example shows.

Example 4.1 *Consider the MDBS in Example 3.3. Suppose that two single-site queries Q_1 ($r_{q_1}(d_1^1)r_{q_1}(d_2^1)$) and Q_2 ($r_{q_2}(d_1^2)r_{q_2}(d_2^2)$) are submitted to servers at S_1 and S_2, respectively directly without informing the GTM, and that both R_1 and R_2 follow two-phase certification protocol. The following execution is then possible:*

$E_1 : r_{r_1}(d_1^1)w_{r_1}(d_1^1)r_{q_1}(d_1^1)r_{q_1}(d_2^1)w_{r_2}(d_2^1)c_{r_1}c_{q_1}c_{r_2}$

$E_2 : r_{r_2}(d_2^2)w_{r_2}(d_2^2)r_{q_2}(d_1^2)r_{q_2}(d_2^2)w_{r_1}(d_1^2)c_{r_2}c_{q_2}c_{r_1}$.

This execution is not serializable, as $R_{1,1}^U$ precedes $R_{2,1}^P$ in E_1 while $R_{2,2}^U$ precedes $R_{1,2}^P$ in E_2.

The ability to process single-site queries as local transactions, without the involvement of the GTM, is critical to data availability and system performance. The use of replicated data is practically motivated by the needs to enhance data availability and to reduce query cost by providing accessible local copies. In [GMW82], the feature, referred to as *insularity*, has been used to characterize transaction processing protocols for both read-only and update transactions.

In the rest of this section, we will discuss the relaxation of the requirement that single-site queries be processed as global transactions. Two alternative approaches are discussed in subsections 4.2 and 4.3. The first approach relaxes the control of the GTM over single-site queries through the use of a weak consistency requirement, the *consistent view* of a database. The second approach does not compromise the consistency requirement and strictly ensures 1SR. It uses a special lock mechanism, called a *propagation lock*, on the primary copy site to avoid any non-1SR execution caused by globally uncontrolled single-site queries.

4.2 Consistent View of Databases

The non-serializability of the execution described above does not necessarily imply that queries Q_1 and Q_2 see an *inconsistent view* of the database. That is, from Q_1's point of view , the database is updated by R_1 followed by R_2, and, from Q_2's point of view, the database is updated by R_2 followed by R_1. In other words, both queries see a database updated by some serial execution of R_1 and R_2.

We say that a query Q in an execution E sees a *consistent view* of the database if (1) the projection of update transactions in E, denoted as $U(E)$, is serializable, and (2) there is no direct conflict cycle (DCC): $Q \prec T_1 \prec T_2 \prec ... \prec T_n \prec Q$ in E, where T_i is an update transaction. The first of these conditions ensures that all updates to the database are consistent, while the second implies that each query is executed in an isolated manner with update transactions.

We further claim that, in the proposed certification approach, single-site queries see a consistent view of the database even if they are processed as local transactions without consulting the GTM. As an illustration, let us consider a single-site query Q which sees an inconsistent view of the database. Thus, either there is a cycle DCC_u in the update transaction projection: $T_1 \prec T_2 \prec ... \prec T_n \prec T_1$ or there is a direct conflict cycle DCC_q: $Q \prec T_1 \prec T_2 ... \prec T_n \prec Q$, where T_i is an update transaction.

Assume first the existence of DCC_q. There are at least two global update transactions in DCC_q, as all local executions are serializable. Eliminating from DCC_q all local transactions (including the query Q), we get the sequence of global update transactions: $T_{g_1}, T_{g_2}, ...T_{g_m}$ where $g_i < g_{i+1}$ ($1 \leq i < m$ and $m \geq 2$).

Since only local transactions may exist between T_{g_i} and $T_{g_{i+1}}$ and T_{g_i} precedes $T_{g_{i+1}}$ for any i ($1 \leq i \leq m$) in DCC_q, there is a k such that T_{g_i} immediately precedes $T_{g_{i+1}}$ in the serialization order O_l^k (where we assume that $T_{g_{m+1}} = T_{g_1}$).

By merging O_l^k for all k, we get a cyclic serialization order O with $T_{g_1} \prec T_{g_2} \prec ... \prec T_{g_m} \prec T_{g_1}$, which contradicts Condition 3.1 of the protocol.

The same result can be obtained for DCC_u.

Thus, we conclude that, in the proposed certification approach, single-site queries always see a consistent view of the database.

The idea of processing single-site queries with a weak consistency requirement is not new. [GMW82] has provided an extensive discussion of this concept and the advantages of providing a consistent view of the database. In this paper, we have applied this formulation to replicated MDBS environments. In

particular, we demonstrate that queries which only read locally available copies see consistent views of the database, if the proposed protocol is used to coordinate the execution of all replica-update and global update transactions.

4.3 Propagation Lock Protocol

In this subsection, we will investigate an alternative approach to processing single-site queries as local transactions. Instead of relaxing the consistency requirement, this approach adopts a *propagation lock* to prevent possible inconsistent accesses.

The purpose of a propagation lock is to block the execution of single-site queries at the primary copy site until the conclusion of other propagation transactions. The lock is set on the primary copy site when a primary-update subtransaction is submitted to the site; it is released after the propagation transaction globally commits at remote sites. The server in the primary copy site is responsible for granting and releasing the locks. Two propagation locks are compatible and can be granted at the same time on the primary copy site. The lock blocks only single-site query transactions, leaving other transactions unaffected. The GTM, which monitors the execution of propagation transactions, will inform the server of the commitment of a propagation transaction.

In this extended approach, a server can submit and commit a single-site query as a local transaction if no propagation lock is set on the site. If a propagation lock is set between the submission and the commitment of a single-site query, the server will abort the query and rerun it after the lock is released. In this way, the single-site query is executed and committed locally without any coordination by the GTM. All other global transactions (including propagation transactions) are still under the control of the certification protocol.

We claim that the two-phase certification protocol, as extended by the propagation lock protocol, will ensure 1SR when the local DBMSs use rigorous concurrency controls.

To show the correctness of the claim, let us first observe that Condition 3.2 is always followed by the extended approach. In fact, for a single-site query at S_i, previously processed as a global transaction G_i, the propagation lock will ensure that this query is committed until all replica-update transactions that precede it have been globally committed. Furthermore, the GTM's certification protocol ensures that any other global transaction G_i or propagation transaction R_i^P will not be committed until all replica-update transactions that precedes it have been globally committed.

Additionally, Condition 3.1 is also followed if local concurrency control produces rigorous schedules. For all global transactions other than single-site queries and for replica-update transactions, the GTM's certification protocol ensures that there exists a total order O over these transactions. This order is compatible with the serialization order O_i^k for all k. Now, consider a set of single-site queries which are under the control of the propagation lock protocol. We claim that these queries do not change the total order at any local site.

It is obvious that queries do not change the order of any of the following transaction pairs: two global transactions, two replica-update transactions initiated from one site, or a global transaction and a replica-update transaction. This is because the orders of these pairs are determined by the GTM's certification protocol and are unaffected by any local executions, including the execution of single-site queries.

These queries also do not change the order of two replica-update transactions which are initiated from different sites if local rigorous schedule is required. Assume these queries change the order at some local site and cause a non-1SR execution (e.g., Example 4.1). In this case, there is at least one query, say Q_1, such that $R_i^U \prec Q_1 \prec R_j^P$ at one site where R_i^U is a primary-update subtransaction and R_j^P is a propagation-update subtransaction. There is another subtransaction or query, say Q_2, such that $R_j^U \prec Q_2 \prec R_i^P$ at another site where R_j^U is the primary-update subtransaction of R_j^P and R_i^P is the propagation-update subtransaction of R_i^U. By the property of local rigorous schedule, which states that transactions commit in an order consistent with their serialization order, R_i^P cannot commit until Q_2 commits; and R_j^P cannot commit until Q_1 commits. By Condition 3.2, Q_2 cannot commit until both R_j^U and R_j^P commit. Thus, R_i^P can not commit until Q_1 commits, which contradicts Condition 3.2.

The proposed propagation lock is similar to the site lock in [AGMS87, BS88] in that both locks take a site as a lock granularity unit. There are, however, some obvious differences in the aspects of lock compatibility and semantics. First, two propagation locks are mutually compatible, and any number of locks can be granted over a site at any time. By contrast, a site lock is not compatible with another site lock. Second, a propagation lock only blocks single-site queries (transactions) and does not block any other transactions. In the site lock method, a site lock will block any global subtransaction.

The propagation lock approach ensures 1SR, a strong consistency requirement. It does not require global control on the execution of a primary-update subtransaction, as the lock is set only on the primary copy site where the primary-update subtransaction is executed. This approach, however, may delay some single-site queries on primary copy sites. It also imposes some restriction on local concurrency control, in the form of rigorous schedules. Nonetheless, when strong consistency is required, the propagation lock offers an attractive approach. The assumption of rigorous schedules is, in general, practically acceptable, as most commercial DBMS products use strict two-phase-locking protocol, which is a rigorous schedule.

5 Conclusion

In this paper, we have proposed a certification-based replicated data management protocol for MDBSs. This protocol employs different replica con-

trol and commit protocols for global and non-global transactions. The main advantage of the protocol is that it allows both deferred updates to replicated copies for local applications and a consistent access to local or nearby copies of a replicated data item by global transactions.

In order for the the GTM to ensure 1SR, local servers must be able to detect the local serialization order of subtransactions when they enter into the PREPARED state. In addition, to ensure the atomicity of replica-update transactions, we further require that the serialization order of two subtransactions be consistent with the order in which they enter into the PREPARED state. We have shown that the requirement can be met through utilizing local rigorous schedules or enforcing local conflicts on tickets.

For the reasons of data availability and system performance, it is desirable that a single-site query which reads non-primary copies be executed locally, without consulting the GTM. We have shown that the certification protocol allows uncontrolled single-site queries to see a consistent view of database, a weak consistent requirement. To strictly ensure 1SR, we also have presented an extension to the basic protocol through the addition of a propagation lock. The extended protocol suffices to avoid any non-1SR execution, even though single-site queries are not controlled by the GTM. However, the use of the propagation lock may delay some single-site queries. The extended protocol works best if the ratio of replica-update transactions to single-site queries (transactions) is small.

Replicated data management in MDBSs is a difficult problem, and many issues remain open. For example, in the proposed protocol, we have assumed the existence of a primary copy for each replicated data item. This, however, may not be possible in some MDBSs. Two local databases containing redundant information may be integrated, and local applications developed independently on each system may then access two copies of the same information. Assigning one of these copies as the primary copy will violate the local autonomy of the other. It is still unclear whether such pre-existent redundant copies can be managed without violating the local autonomy of both local systems.

6 Acknowledgements

The second author would like to thank Ming-Chien Shan and Marie-Anne Neimat from HP labs. for their comments and suggestions on the original manuscript. We would also like to thank the referees for their helpful comments.

References

[AGMS87] R. Alonso, H. Garcia-Molina, and K. Salem. Concurrency control and recovery for global procedures in federated database systems. In *IEEE Data Engineering Bulletin*, 10(3), 1987.

[BHG87] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publishing Co., 1987.

[BGRS91] Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz, and A. Silberschatz. On rigorous transaction scheduling. In *IEEE Transactions on Software Engineering*, 17(9), 1991.

[BGS92] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. In *International Journal on Very Large Data Bases.* 1(2), 1992.

[BS88] Y. Breitbart and A. Silberschatz. Multidatabase Update Issues. In *Proceedings of the ACM International Conference on Management of Data*, 1988.

[CL91] M. Carey and M. Livny. Conflict Detection Tradeoffs for Replicated Data. In *ACM Transactions on Database Systems*, 16(4), 1991.

[DEKB93] W. Du, A. Elmagarmid, W. Kim, and O. Bukhres. Supporting consistent updates in replicated multidatabase systems. In *International Journal on Very Large Data Bases*, 2(2), 1993.

[GMW82] H. Garcia-Molina and G. Wiederhold. Read-only transactions in a distributed database. In *ACM Transactions on Database Systems*, 7(2), 1982.

[GRS91] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. On serializability of multidatabase transactions through forced local conflicts. In *Proceedings of IEEE International Conference on Data Engineering*, Kobe, Japan, 1991.

[RAZ92] Y. Raz. The Principle of Commitment Ordering, or Guaranteeing Serializability in a Heterogeneous Environment of Multiple Autonomous Resource-Managers. In *Proceedings of International Conference on Very Large Data Bases*, 1992.

[Ston79] M. Stonebraker. Concurrency control and consistency of multiple copies of data in distributed INGRES. In *IEEE Transaction on Software Engineering* 3(3), 1979.

[WQ90] G. Wiederhold and X. Qian. Consistency Control of Replicated Data in Federated Databases. In *Proceedings of Workshop on Management of Replicated Data*, Houston, Texas, 1990.