Deferred Updates and Data Placement in Distributed Databases [1]

Parvathi Chundi *Daniel J. Rosenkrantz* *S. S. Ravi*
Department of Computer Science
University at Albany - State University of New York
Albany, NY 12222

Abstract

Commercial distributed database systems generally support an optional protocol that provides loose consistency *of replicas, allowing replicas to be inconsistent for some time. In such a protocol, each replicated data item is assigned a primary copy site. Typically, a transaction updates only the primary copies of data items, with updates to other copies* deferred *until after the transaction commits. After a transaction commits, its updates to primary copies are sent transactionally to the other sites containing secondary copies. We investigate the transaction model underlying the above protocol. We show that global serializability in such a system is a property of the placement of primary and secondary copies of replicated data items. We present a polynomial time algorithm to assign primary sites to data items so that the resulting topology ensures serializability.*

1 Introduction

A widely used method for improving the reliability and availability of data in distributed databases is replication of data items. By storing critical data at multiple sites, the system can continue to operate even when some of the sites fail. Also, performance can be improved by accessing the nearest copy. An important goal of concurrency control for a replicated database system is achieving replica transparency. That is, a replicated database system should behave like a single copy database as far as the users are concerned. Therefore, the interleaved execution of transactions on a replicated database should be equivalent to a serial execution of these transactions on a single copy database. Transaction processing in replicated database systems has been studied by a number of researchers (see for example [AE90, AE92, BHG87, BG84, BL93, PL91] and the references contained therein). Several protocols achieving replica consistency, such as two phase commit and quorum consensus, have been proposed in the literature [BL93, BHG87, Gi79, Th79]. These protocols, which we refer to as **multi-site commit protocols**, allow a transaction to access a replicated data item at a site only if a certain subset or a specified number of replicas of that data item agree. Almost all commercial distributed database system products available today provide two phase commit as an option [Sc94]; the advantage of utilizing this option is that it guarantees the consistency of replicated data and the serializability of transactions. In a two phase commit approach to updates, a transaction that updates a replicated data item is committed only after *all* the sites containing a copy of that data item agree to commit the transaction. This approach is often referred to as **tight consistency** or **synchronous distributed replication**. Tight consistency has some drawbacks in practice [Mo94, MP+93, Sc94]. Many database vendors indicate that in numerous applications, two phase commit is impractical. For instance, quoting from [MP+93],

> "Synchronous distributed replication is attractive in theory, but fails in the real world. Its reliance on 100 percent system availability makes maintaining a productive level of transaction throughput for distributed replication impossible."

An overview of commercially available distributed database systems with replication capability appears in [Go94]. Several of these systems, including SYBASE System 10 [Mo94, Co93, MP+93], Oracle 7 [Or93], IBM Datapropagator Relational [Ib94], and CA-OpenIngres [Sc94], support a multi-site protocol for tight consistency and also include an *optional* protocol that *defers* the update of replicas. The latter deferred update protocols support **loose consistency** [Mo94, MP+93] by allowing some of the copies to be inconsistent for some time. However, loose consistency provides better responsiveness since the waiting opera-

[1] Research Supported by NSF Grant CCR-90-06396. Email addresses: {paru, djr, ravi}@cs.albany.edu

tions associated with multi-site commit protocols are avoided. A similar approach, called **asynchronous coherency control**, has been been studied in [PL91].

When a **deferred update** is used, a transaction can commit after updating only one copy of a replicated data item. After the transaction commits, the update is propagated asynchronously to the other copies. There are several approaches that achieve deferred update. We focus on the **primary copy** approach. In this approach, each replicated data item is assigned a site where the primary copy of the data item is stored. Copies of this data item appearing at other sites are referred to as **secondary copies**. Different data items may have primary copies at different sites and a given data item need not have a copy at every site. (The notions of primary and secondary copies have also been used in Distributed INGRES [St79, SN77].) A **replication server**[2] is included at each site; the replication servers co-operate to ensure the propagation of an update to all copies.

This paper deals with a form of primary copy approach which we call the **strict primary copy** approach. In a **strict primary copy** system, the updates to a data item are done at its primary site and the secondary copies are read-only. For a transaction T to update a replicated data item, T must operate at the site S containing the primary copy. To commit, T must commit at S. Once T commits, the replication server at S sends all updates by T on primary copies at S transactionally to all sites containing corresponding secondary copies. Since the commit of a transaction updating a primary copy at a site results in messages going to other sites, we call such an update message a **ripple**. We call the subtransaction executed as a consequence of receiving a ripple a **ripple subtransaction**. Thus, the values of the primary and secondary copies of a data item may differ until the updates on the secondary copies are completed. Spawning various activities at different sites to maintain the consistency of interdependant data, depending on a transaction updating a data item at a single site is also discussed in [SRK92].

For many applications, use of a deferred update protocol seems preferable to two phase commit. Use of the strict primary copy approach prevents update conflicts. However, a further goal might be to ensure that histories of transaction executions be serializable. A discussion of the importance of serializability in deferred update systems and examples of nonserializable executions in such systems can be found in [Go95]. References [Mo94, MP+93, Or93] provide sample applications with simple topologies (e.g. a star) of primary and secondary copies which seem to work fine, although the issue of serializability is not addressed. For example, consider a commercial organization with a headquarters and branch offices. The primary copy of the database is located at the headquarters site. Each branch office needs to view that data which pertains to its operations. Therefore, data is replicated at the branch offices, but the updates can be initiated only at the headquarters. These updates are forwarded to the branch offices transactionally. This replication topology is a star where the headquarters site is the center and the branch offices are the leaves. In another example using a star topology [MP+93], each branch office has the primary copy of its data, and the headquarters site has a secondary copy. In these examples, transactions operate at only one site using a strict primary copy approach, with updates propagating asynchronously to secondary copies.

In this paper, we investigate serializability[3] of transaction processing systems utilizing a primary copy deferred update approach. Perhaps, the most restricted system design that can be said to use "distributed transaction processing" is one which uses a strict primary copy approach, and where each transaction can operate *at only one site*. (Once a transaction commits at the site where it operates, ripples are sent asynchronously to sites with secondary copies of the data items updated by the transaction.) We formalize a protocol, referred to as the **strict primary update (spu)** protocol, that utilizes this constrained transaction model.

In this paper, we show that even for the very restricted spu-protocol, the topology of data distribution among sites must satisfy certain constraints in order to ensure serializability. In fact, it is easy (see Section 2) to construct examples of non-serializable global histories under this model. We develop a tight characterization based on the topology of data distribution. We define a directed graph (called the **data placement graph**) which represents the distribution of the primary and secondary copies of data items across the sites. Under standard assumptions concerning the nature of the distributed database system and the concurrency control mechanisms used at each site, we show that global serializability is ensured if and only if the data placement graph satisfies an acyclicity condition. More specifically, for any configuration violating the acyclicity condition, a non-serializable

[2] The term "replication server" is a trademark of SYBASE, Inc.

[3] Throughout this paper, we use the term "serializability" to refer to conflict serializability [BHG87].

history can be produced involving transactions that use the strict primary copy approach and where each transaction operates at only one site. This is a very strong lower bound because it applies to even the most restricted kind of transactions.

We also present an efficient algorithm for the following primary site assignment problem: Given a list of sites for each data item (the list specifies the set of sites at which the data item is placed), choose a primary site for each data item so that the resulting data placement graph satisfies the acyclicity condition mentioned above. This problem arises when an application needs an initial assignment for its data items. A slightly different form of the problem arises when one or more sites containing primary copies of some data items fail, and for each such data item, one of the secondary copy sites is to be redesignated as the primary copy site without violating the acyclicity condition.

When transactions can modify only the primary copies of data items, the primary site assignment may determine whether certain transactions can be executed using the spu-protocol. In this case, we need to determine if there is a primary site assignment that both satisfies the acyclicity condition, and for which each transaction type can be assigned to a site where it can operate. For this problem, we are again given a list of data items (each accompanied by a list of sites where it is located) and a list of transaction types (each accompanied by a read and write set of data items). We present an efficient algorithm that finds a feasible primary site assignment for each replicated data item and an operating site for each transaction, whenever a solution exists.

The SDD-1 system [BSR80, BS80, RB+80] used the idea of preanalyzing transaction classes to identify access conflicts between transaction classes and to establish which synchronization mechanism to use for each such conflict. In this paper we assume that each site's concurrency control uses a standard mechanism, such as two-phase locking, to handle all conflicts, so such a preanalysis is not needed. The transaction class analysis that we consider assigns transactions to sites, and so addresses a different problem than considered in the SDD-1 system.

The concept of preanalyzing transaction classes is also used in [SW84] to increase availability and maintain database consistency when a distributed database system with replication becomes partitioned. The issues considered in this paper are different from those considered in [SW84] because we do not address failures.

The remainder of this paper is organized as follows. Section 2 presents a non-serializable scenario that occurs even with simple transactions. Section 3 presents the distributed data model and introduces related definitions. Section 4 formalizes the spu-protocol and presents the characterization of serializability in a system executing the spu-protocol. Section 5 presents algorithms for the primary site selection and transaction assignment problems. Section 6 concludes the paper.

2 Nonserializable Example

A distributed and replicated database system must ensure **replica consistency** as well as **database consistency**. When one copy of a replicated data item is updated, to achieve replica consistency, all other copies must be modified to reflect this change. Database consistency is ensured by permitting only serializable histories in the system. If two transactions are allowed to update two copies of a replicated data item located at two different sites without consulting each other, the resulting execution contains update conflicts and is non-serializable [4]. The strict primary approach supported in various commercial systems such as Sybase System 10, Oracle 7, and IBM Datapropagator Relational, etc. ensures replica consistency and avoids update conflicts. However, it is not clear whether serializability is guaranteed in general in a system using the spu-protocol. The vendors do not provide any guidelines as to how to configure the system so that database consistency is guaranteed when the spu-protocol is used. We now present applications that use a deferred update approach where serializability is an important goal. Nevertheless, the spu-protocol may produce non-serializable histories. In the following examples, $r_i(x)$ and $w_i(x)$ denote the read and write operations of a transaction T_i on a data item x and $send_i$ and $recvd_i$ denote the send and receive operations of a ripple generated as a result of write operation of T_i on some primary data items. Finally, c_i denotes the commit operation of T_i.

Example 2.1: Consider a manufacturing company with three sites where S represents the sales office site, P represents the production site and A represents the central administration site (refer to Figure 2.1). The sales office accepts purchase orders and informs the production site so that it can service the purchase order. The sales office also sends this purchase order information to the central administration site for record keeping. Hence, the data item *purchase_order* is replicated at all three sites, with the primary copy at S.

[4] Such an execution may need human intervention or special rules to resolve the conflict.

The production site records the purchase order information. This information is used to increase production of this item when the production site next revises its schedule. The production site then informs the central administration of the new schedule of production. Hence, the data item *production_quantity* is replicated at P and A with the primary copy at P. The central administration site A stores the information from the sales office as well as from the production site. Site A also decides the asking price of an item produced based on how much of it is scheduled to be produced, and how much of this production is accounted for by purchase orders. If there is a large surplus of items scheduled to be produced, a lower price is set. If the scheduled over-production of the item is small (i.e., the item is in demand compared to supply), a high price is set.

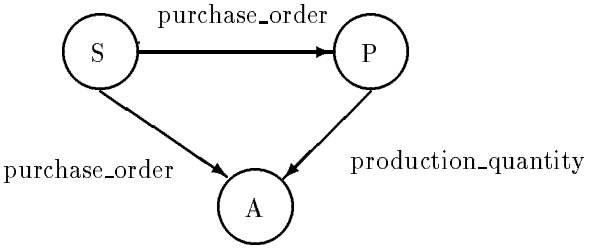

Figure 2.1

Now, consider the following scenario. A transaction T_s operating at site S records a purchase order, thereby updating the primary copy of the data item *purchase_order*. Hence, after T_s commits, a ripple is sent both to P and A to update the secondary copies. The site P receives this message and updates the secondary copy of *purchase_order*. A transaction T_p at P reads this value and writes a new value for the primary data item *production_quantity*. At site P, T_p reads a value written by T_s. Therefore, the local serialization graph at P contains the edge $T_s \rightarrow T_p$. Since the primary copy of *production_quantity* is updated, a ripple is sent to the site A to update the corresponding secondary copy.

Hence, both sites S and P send update messages to A. Suppose site A receives the ripple corresponding to T_p from P first[5]. It updates the secondary copy *production_quantity* at A. Immediately after this, suppose a local transaction T_a analyzes the database in order to post an asking price for the item. After T_a reads the database and commits, the ripple corresponding to T_s is received from S and the new value for *purchase_order* is written at A. Hence, at A, T_a read an after-value of T_p and a before-value of T_s. Hence, there is a path $T_p \rightarrow T_a \rightarrow T_s$ in the local serialization graph of A.

Hence, the cycle $T_s \rightarrow T_p \rightarrow T_a \rightarrow T_s$ is created when the serialization graphs from all three sites are merged. To see how this cycle may lead to inconsistencies, note that the analysis by T_a shows a high production quantity for the item requested by T_s(updated by T_p) and a low number accounted for by purchase orders (the ripple corresponding to T_s had not arrived at A yet.). This may lead to A setting a lower price for the item. This price may not be consistent with the price that A would have selected had the ripple from S arrived before T_a read the database, i.e., the final values of price, production quantity and purchase order total are inconsistent. Hence, the non-serializable execution may lead to a significant loss. □

[5]This can happen if the message from S is delayed due to network traffic, or if S is farther away from A than P.

3 Data Placement

3.1 Distributed Data Model

A replicated database system consists of a set of **sites** S_1, S_2, ..., S_n. Every pair of sites communicates using a fifo discipline. Each site contains a **protocol manager** that communicates with other sites to maintain the consistency of the replicated data. Each replicated data item has a designated site where the **primary copy** of the data item resides; the copies of this data item at other sites are **secondary copies**. Data items occurring at more than one site are referred to as **global data items**. Each site may also have some **local data items**. We denote a secondary copy of a data item d by d' and the primary copy by d itself.

3.2 Data Placement Graph

We define a directed graph called the **data placement graph (DPG)** as follows. Each node in the DPG represents a site. There is a directed edge from S_i to S_j if there is at least one data item for which S_i is the primary site and S_j is a secondary site. Note that a DPG cannot contain a self loop. Given a DPG, the corresponding **undirected data placement graph (UDPG)** is obtained by simply erasing the directions on the edges and combining multi-edges (if any) between a pair of vertices into a single edge.

Finally, we mention some graph theoretic concepts used in this paper. In a directed graph, a pair of edges of the form (u, v) and (v, u) which form a directed cycle of length 2 will be referred to as **dual edges**. We say that a directed graph $D(V, A)$ is **strongly acyclic** if it does not have dual edges and the undirected graph obtained from D by deleting the direction on each edge and combining the multi-edges into a single edge is acyclic.

A **source** node of a directed graph is a node with no incoming edges, and a **sink** node is a node with no outgoing edges.

4 Spu Protocol and Database Consistency

4.1 Description of the Protocol

Under the spu-protocol, each transaction *operates* at only one site and gets committed at that site. (If a transaction updates primary copies of the global data items, the ripples are propagated to secondary sites after it commits.) A transaction is a partial order of read and write operations (denoted by r and w respectively) ending with a commit (c) or an abort (a) operation [BHG87]. The read (write) operation of a transaction T_i on a data item d is denoted by $r_i(d)$ ($w_i(d)$). Further, c_i and a_i denote T_i's commit or abort operation respectively.

At any site, the spu-protocol can execute the following two types of transactions:

1. A **local** transaction can read and write local data items at that site and can read global data items at that site (but cannot write global data items).

2. An **pg** (primary-global) transaction can read and write local and primary data items at that site. Further, the transaction can read, but not write, secondary data items at that site.

Note that a local transaction is a special case of a pg transaction.

The send and receive operations of a ripple generated by the execution of a pg transaction T_i are denoted by $send_i(\Delta)$ and $recvd_i(\Delta)$ respectively, where Δ contains the new values of primary copies modified by T_i. The ripples are sent in the order in which the pg transactions committed at S. Also, for each sending site S_j, the protocol manager at a receiving site submits the ripple subtransactions in the order in which the corresponding ripples arrived from S_j.

We define the local history at a site as a partial order over the operations of all transactions executing at that site, including send and receive operations carried out by the protocol manager at that site. The partial order at each site must satisfy the usual conditions from [BHG87]. A global history is the union of local histories at each site, with the partial order augmented by the requirement that the send of a ripple occurs prior to the corresponding receive operation. We define an **spu-global history** to be a global history that can be produced by the spu-protocol. The **global serialization graph (GSG)** is obtained by taking the union of nodes and edges of the local serialization graph at each site. In the GSG, the node corresponding to a ripple subtransaction (if any) is identified with the node which caused this ripple. A global history is serializable if and only if its GSG is acyclic.

We assume that each local database concurrency control produces only serializable histories and that the local histories have the property that the commit order of transactions is a serialization order. (This can be ensured by using an appropriate concurrency control mechanism, such as two-phase locking, at each local database.) We also assume that a ripple subtransaction is always successfully committed. If it gets aborted, it is retried.

One of the main contributions of this paper is a characterization of those DPGs for which the spu-protocol always produces serializable global histories. We define a DPG to be **spu-global serializable** if it has the property that every history produced by the spu-protocol operating in a data configuration corresponding to the DPG is globally serializable. Note that spu-global serializability is a property of a DPG.

4.2 Characterization of Consistency

In this section, we discuss how spu-global serializability can be achieved in a replicated distributed database system running the spu-protocol at each site. It is easy to construct a scenario in the above system in which each local history is serializable, but the resulting global history is not serializable (refer to Section 2). In Example 2.1 (Figure 2.1), the order in which ripples from sites S and P arrive at A produces a non-serializable execution. Note that the DPG in Example 2.1 is not strongly acyclic. This example illustrates that the structure of a DPG may play a role in determining when global serializability is guaranteed under the spu-protocol. We prove the conditions under which global serializability can be guaranteed using the following theorem. We omit the proof of this theorem due to space limitations.

Theorem 4.1 *A DPG is spu-global serializable if and only if it is strongly acyclic.* □

Theorem 4.1 proves that the strong acyclicity condition is required for ensuring serializability even when the transactions in the system are only local and primary-global. Since any implementation of the primary-copy approach will contain these types of transactions, this condition applies to any such implementation.

5 Selecting Primary Sites and Assigning Transactions

5.1 Overview

In this section, we first present an efficient algorithm that determines for a given data distribution, whether there is an assignment of a primary site to each data item so that the resulting DPG is acyclic; if yes, the algorithm outputs such an assignment. We then consider this problem in the context of a given set of transaction classes, each of which is to be implemented as a pg transaction. Since update operations on global data items can be done only at their primary sites for such transactions, the placement of primary copies may affect the executability of some transactions. Therefore, while assigning the primary sites to data items, we need to also take into consideration the access sets (i.e., read and write sets) of transaction classes to be executed in the system. We show how to modify the initial algorithm to take into account the access sets of transactions in assigning the primary sites and find a solution (if one exists) such that each data item is assigned a primary site, the resulting DPG is strongly acyclic, and each transaction is assigned a site which contains copies of all the data items in its read set, and primary copies of all the data items in its write set.

5.2 Selecting Primary Sites for Data Items

Theorem 4.1 points out that the structure of the DPG plays a crucial role in determining the serializability of the global histories produced by executing the spu-protocol at each site. Thus, given a distribution of data items across the sites, it is necessary to select primary sites for each replicated data item so that the resulting DPG is strongly acyclic. We call an assignment of primary sites to data items **valid** if the resulting DPG is strongly acyclic. It is easy to construct examples of data distribution for which there is no valid assignment. In this section, we address the question of determining whether there is a valid assignment for a given data distribution, and if so, producing one such assignment. We present a polynomial time algorithm for this problem, which we refer to as the **primary site selection (PSS)** problem.

Primary Site Selection: (PSS)

Instance: A set $S = \{s_1, s_2, \ldots, s_n\}$ of sites, a collection $D = \{d_1, d_2, \ldots, d_m\}$ of data items, and a **site set** $\Phi_i \subseteq S$ for each data item d_i (i.e., Φ_i is the set of sites containing data item d_i), $1 \leq i \leq m$.

1. Construct the auxiliary graph Γ for the given instance I of the PSS problem.
2. Find the connected components $\Gamma_1, \Gamma_2, \ldots, \Gamma_r$ of Γ.
3. **for** $j := 1$ **to** r **do**

   (i) Let $d_{i_1}, d_{i_2}, \ldots, d_{i_p}$ be the data item in Γ_j.

   (ii) Compute $\Psi = \cap_{l=1}^{p} \Phi_{i_l}$.

   (iii) **if** Ψ is empty
   then Print "No Solution" and stop
   else Choose an arbitrary site from Ψ as the primary site for all the data items in Γ_j.
4. Construct the data placement graph G for I using the primary site assignments chosen in Step 3.
5. **if** G is strongly acyclic **then** Print the assignment found in Step 3 **else** Print "No Solution".

Figure 5.1: Description of Algorithm-PSS

Requirement: Determine whether there is an assignment of a primary site to each data item d_i such that the resulting DPG is strongly acyclic. If yes, find one such assignment.

In the remainder of this section, we outline a polynomial algorithm for the PSS problem. We begin with a lemma which points out an important constraint that must be satisfied by all valid assignments. The proof of the lemma is straightforward.

Lemma 5.1 *Let I be an instance of the PSS problem. Suppose d_i and d_j are data items which occur together in two or more sites (i.e., the corresponding site sets Φ_i and Φ_j satisfy the condition $|\Phi_i \cap \Phi_j| \geq 2$). Then, in any valid assignment for I, d_i and d_j must have the same primary site.* □

Given an instance I the of PSS problem with data set $D = \{d_1, d_2, \ldots, d_m\}$ and the corresponding site sets $\Phi_1, \Phi_2, \ldots, \Phi_m$, we define the **auxiliary graph** Γ as follows. Γ is an undirected graph with a node for each data item. There is an edge $\{d_i, d_j\}$ in Γ if data items d_i and d_j occur together in two or more sites. The role played by the auxiliary graph is indicated in the following lemma, which is a consequence of Lemma 5.1 and the definition of auxiliary graph.

Lemma 5.2 *Let I be an instance of the PSS problem and let Γ denote the corresponding auxiliary graph. Suppose Γ' is a connected component of Γ. In any valid assignment to I, all the data items in Γ' must have the same primary site.* □

The steps of our algorithm (Algorithm-PSS) are shown in Figure 5.1. Whenever the algorithm outputs an assignment, it is clear from Figure 5.1 that

the assignment is valid. From Lemma 5.2, it can be verified that if the algorithm outputs "No Solution" in Step 3(iii), then the given instance of PSS does not have a valid assignment. Using a more involved argument, it can be shown that when the algorithm outputs "No Solution" in Step 5, the given instance does not have a valid assignment. Thus, we can conclude that given any instance of the PSS problem, Algorithm-PSS correctly produces a valid assignment whenever such an assignment exists.

The following example illustrates the above algorithm.

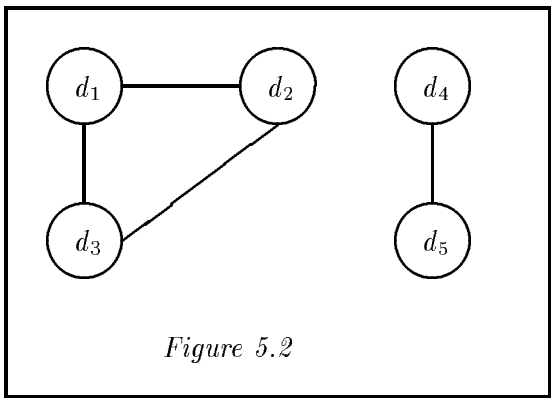

Figure 5.2

Example 5.1 The set of sites Φ_i at which each data item d_i occurs are as follows. Φ_1: $\{S_1, S_2, S_3, S_4\}$, Φ_2: $\{S_1, S_2, S_4\}$, Φ_3: $\{S_1, S_3, S_4\}$, Φ_4: $\{S_4, S_5\}$ and Φ_5: $\{S_4, S_5, S_6\}$.

The auxiliary graph for this instance of PSS (Figure 5.2) has two connected components; one component Γ_1 containing d_1, d_2, and d_3 and the other component Γ_2 containing d_4 and d_5. The intersection set Ψ for Γ_1 is $\{S_1, S_4\}$. We choose S_1 arbitrarily as the primary site for data items d_1, d_2 and d_3 in Γ_1. The connected component Γ_2 has the intersection set $\Psi = \{S_4, S_5\}$. The site S_5 is arbitrarily chosen to be the primary site for data items in Γ_2, i.e., d_4 and d_5. The resulting DPG is strongly acyclic. Hence, the algorithm found a valid assignment. □

5.3 Assigning Transactions to Sites

We briefly discuss an algorithm that produces an assignment of transactions to sites, along with an assignment of primary sites to data items, if such assignments are possible. We base this algorithm on the algorithm given in Figure 5.1. In addition to the data item set D and the corresponding site sets, the algorithm takes as input a set of pairs $\mathcal{P} = \{\langle RS_k, WS_k\rangle \mid T_k$ is a transaction in the system$\}$ where RS_k and WS_k respectively denote the read set and write set of transaction T_k.

We first construct the auxiliary graph Γ for the given data distribution. We then introduce additional edges into Γ as follows. We add an edge $\{d_i, d_j\}$ if there is a transaction T_k such that $\{d_i , d_j\} \subseteq WS_k$. Let Γ' be the resulting graph. Let Γ_1, Γ_2, ..., Γ_r be the connected components of Γ'. For each Γ_j, we construct the set Ψ of sites containing all the data items in Γ_j. If Ψ is empty, we conclude that there is no solution. Otherwise, let GT_j be the set of transactions T_k such that $WS_k \subseteq \Gamma_j$. For each site S_i in Ψ, we check for each transaction T_k in GT_j, whether all data items from RS_k and WS_k are present at S_i. If there is no site for which this condition is true, we conclude that there is no solution. Otherwise, an arbitrary site S_i satisfying the condition is chosen to be the primary site for all the data items in Γ_j, and all transactions in GT_j are assigned to S_i.

We then check whether the above assignment produces a strongly acyclic DPG. If not, the above instance does not have a solution. Otherwise, we have a solution that assigns primary sites for data items and also assigns the given set of transactions to sites such that their access sets are satisfied. Clearly, the algorithm runs in polynomial time.

6 Conclusions

A deferred update approach is supported by several commercial database systems, such as Sybase System 10, Oracle 7, CA-OpenIngres, and IBM Datapropagator Relational etc., to maintain replica consistency efficiently. In this paper, we focussed on the strict primary copy approach. Even though the strict primary copy approach avoids update conflicts, the issue of when this approach guarantees serializability has not been addressed in the literature. We formalized the strict primary update protocol and developed a tight characterization of serializability based on placement of primary and secondary copies. We also presented efficient algorithms for selecting primary sites and for assigning transactions to sites.

The spu-protocol is *basic* to any implementation of the primary copy approach in the sense that it will be embedded as a special case within any more general protocol. Therefore, strong acyclicity of the data placement graph is a necessary condition for maintaining database consistency in any such system. Several applications where deferred update is applicable are discussed in [Mo94, Co93, MP+93, Or93]. Interestingly, the data placement graphs used in all of those examples are strongly acyclic. This suggests that using a strongly acyclic data placement graph will be feasible in many practical situations.

References

[AE90] D. Agrawal and A. El Abbadi, "Exploiting Logical Structures of Replicated Databases," *Inf. Proc. Lett.*, Vol. 33, No. 5, Jan. 1990, pp 255-260.

[AE92] D. Agrawal and A. El Abbadi, "The Generalized Tree Quorum Protocol: An Efficient Approach to Managing Replicated Data," *ACM TODS*, Vol. 17, No. 4, Dec. 1992, pp 689-717.

[BG84] P. A. Bernstein and N. Goodman, "An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases," *ACM TODS*, Vol. 9, No. 4, Dec. 1984, pp 596-615.

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

[BSR80] P. A. Bernstein, D. W. Shipman, and J. B. Rothnie, Jr., "Concurrency Control in a System for Distributed Databases (SDD-1)," *ACM TODS*, Vol. 5, No. 1, March 1980, pp. 19-51.

[BS80] P. A. Bernstein, D. W. Shipman, "The correctness of Concurrency Control Mechanisms in a System for Distributed Databases (SDD-1)," *ACM TODS*, Vol. 5, No. 1, March 1980, pp. 52-68.

[BL93] A. J. Bernstein and P. M. Lewis, *Concurrency in Programming and Database Systems*, Jones and Bartlett Publishers, Boston, MA, 1993.

[Co93] M. Colton, "Replicated Data in a Distributed Environment," *Proc. 1993 ACM SIGMOD Conf.*, Washington, DC, May 1993, pp 464-466.

[Gi79] D. K. Gifford, "Weighted Voting for Replicated Data," *Proc. 7th SOSP*, Dec. 1979, pp 150-159.

[Go94] R. Goldring, "A discussion of Relational Database Replication Technology," *InfoDB*, Vol.8, No.1, Spring 1994.

[Go95] R. Goldring, "Update Replication: What Every Designer Should Know," *InfoDB*, Vol.9, No.2, Apr. 1995, pp. 17-24.

[GR93] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan-Kaufmann Publishers, San Mateo, CA, 1993.

[Ib94] IBM, *An introduction to Datapropagator Relational*, Release 2, Technical Document, IBM, Dec 1994.

[Mo94] A. Moissis, "SYBASE Replication Server: A Practical Architecture for Distributing and Sharing Corporate Information," Technical document, SYBASE Inc., March 1994.

[MP+93] N. Monserrat, T. Palanca, M. Deppe and B. Hartman, "Replication Server: A Component of SYBASE System 10," Technical document, SYBASE Inc., April 1993.

[Or93] Oracle Corporation, "Oracle 7^{TM} Symmetric Replication: Asynchronous Distributed Technology," White paper, Sept. 1993.

[PL91] C. Pu and A. Leff, "Replica Control in Distributed Systems: An Asynchronous Approach," *Proc. 1991 ACM SIGMOD Conf.*, Denver, CO, May 1991, pp 377-386.

[RB+80] J. B. Rothnie, Jr., P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong, "Introduction to a System for Distributed Databases (SDD-1)," *ACM TODS*, Vol. 5, No. 1, March 1980, pp 1-17.

[SRK92] A. Sheth, M. Rusinkiewicz, and G. Karabatis, "Using Polytransactions to Manage Interdepedent Data," *Transaction Models for Advanced Database Applications*, Ed. A. Elmagarmid, Morgan-Kaufmann, 1992.

[SW84] D. Skeen and David D. Wright, "Increasing availability in Partitioned Database Systems," *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Waterloo, Canada, Apr. 1984, pp 290-299.

[Sc94] G. Schussel, "Database Replication: Playing Both Ends Against the Middleware," *Client/Server Today*, Nov.1994, pp 57-67.

[St79] M. Stonebraker, "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," *IEEE Trans. Soft. Engg.*, Vol. SE-5, No. 3, May. 1979, pp 188-194.

[SN77] M. Stonebraker and E. Neuhold, "A Distributed Database Version of INGRES," *Proc. 2nd Berkeley Workshop on Distributed Databases and Computer Networks*, Berkeley, CA, May 1977, pp 19-36.

[Th79] R. H. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases", *ACM TODS*, Vol. 4, No. 2, June 1979, pp 180-209.