Journal of Electronics & Computer Science, Vol.12, No.1, 2010

Consistent Execution of Concurrent Transactions in Peer Data Sharing Systems

Md Mehedi Masud, Sultan Aljahdali

Department of Computer Science, Taif University, Taif, Saudi Arabia.
{mmasud, aljahdali}@tu.edu.sa

Abstract: In this paper, we investigate the mechanisms of transaction processing in a peer data sharing system. A peer data sharing system is a collection of autonomous data sources, called peers, where each peer augments a conventional database management system with an interoperability layer (i.e. mappings) for sharing data and services. In this network, each peer independently manages its database and can execute queries as well as updates over the related data in other peers. For transaction processing mechanism, this paper focuses on the correct execution of concurrent transactions in a peer data sharing system. A correctness criterion is introduced to ensure the consistent execution of concurrent transactions in the system. In order to guarantee correctness, the paper proposes an approach, called Enforced Serialization Order.

Keywords: database, transaction processing, peer to peer

1. Introduction

In a P2P system, all participating computers (or peers) have compatible capabilities and responsibilities, and exchange resources and services through pair-wise communication, thus eliminating the need for centralized servers. Until now, there are many domain specific P2P systems (e.g. Freenet, Gnutella, SETI$@$home, ICQ, etc.) that have already been deployed. With a few notable exceptions, currently implemented P2P systems lack data management capabilities that are typically found in a database management system (DBMS), such as query and transaction processing.

A peer data sharing system (PDS) combines both P2P and database management system functionalities. The local databases on peers are called *peer databases*. In a PDS, each peer chooses its own data model and schemas, and maintains data independently of other peers. Contrary to the traditional data integration systems where a global mediated schema is required for data exchange, in a PDS, the semantic relationships exist between a pair of peers, or among a small set of peers. Any peer can contribute new data, schema information, or schema mappings with other peers' schemas. The data can be shared globally among the peers by traversing the transitive relationships among semantically related peers.

In the last few years, steady progress has been made in research on various issues related to PDSs, such as data integration model, mediation methods [1, 2] coordination mechanisms [3, 4, 5] and data-level mappings [6] among databases in peers. However, the vast majority of the literature on PDSs has focused on query processing, since, intuitively, data sharing in a PDS occurs through query translation and propagation across the network.

In this paper we investigate the execution of transactions in a data sharing system. We observe that transaction processing in a data sharing system is similar to processing in a multidatabase system (MDBS) [7, 8] in the sense that each system consists of a collection of independently created local database systems (LDBSs), each of which may follow different concurrency control mechanisms. Moreover, each system supports the management of transactions at both local and global levels. Global transactions are those that execute at several sites and local transactions are those that execute at a single site. In an MDBS, global level transactions

are issued to the global transaction manager (GTM), and are decomposed into a set of global subtransactions to be individually submitted to the corresponding LDBSs.

However, in a peer data sharing system, a global transaction is not decomposed, but rather propagated as an entire transaction (that is, not the individual read and write operations that constitute the transaction). The remote peer that receives the transaction considers the transaction as submitted by local users. In MDBSs, transactions are executed under the control of the GTM. One of the main problems in MDBSs is ensuring serializability of the global schedule under the assumption that local schedules at each LDBS are serializable.

A schedule S_k in a site s_k is a sequence of operations resulting from the execution of transactions located on that site. A serialization graph for a schedule S_k is a directed graph with nodes corresponding to the transactions that are committed in S_k and with a set of edges such that $T_i \rightarrow T_j$ if T_i conflicts with T_j in S_k. In what follows, $o_i(x)$ denotes an operation o_i of transaction T_i on database object x; $w_i(x)$ and $r_i(x)$ denote write and read operations, respectively. A transaction T_i and T_j are in direct conflict in schedule S_k, denoted by $T_i \rightarrow T_j$, at site s_k if and only if there exists operations $o_i(x)$ in T_i and $o_j(x)$ in T_j, $T_i \neq T_j$, such that $o_i(x)$ followed by $o_j(x)$, where one of them is a write operation on a data item x and T_i does not abort before $o_j(x)$ is executed [8]. A schedule S_k is serializable if and only if the serialization graph is acyclic [9]. A global schedule **S** is a partial ordered set of all operations belonging to local and global transactions such that, for any local site s_k, a projection of **S** on a set of global and local transactions executing at site s_k is the local schedule S_k at site s_k [8]. Global serializability is ensured if there exists a total order defined over global transactions that is consistent with the serialization order of global transactions at each of the LDBSs. Since global transactions are under the control of the global transaction manager in MDBSs, the global transactions' serializability is enforced by a GTM.

In contrast, a data sharing system is built on a network of peers without a global transaction manager or controller. However, we can assume that each LDBS ensures the local serializability using the local concurrency protocol. Since the execution of concurrent transactions in a data sharing system is similar to the execution in an MDBS, we also consider the concurrent transactions' execution consistency criteria as ensuring serializability.

The absence of a GTM in a data sharing system makes it more challenging to ensure global serializabilty since peers execute transactions independently beyond any centralized control.

We observe that, although we assume that the LDBS of each peer guarantees serializability, concurrent transactions that execute in multiple peers may be serialized in different orders in different peers, resulting in an inconsistent execution of the transactions in the system.

1.1 Motivating Example

Consider a data sharing environment consisting of three peers: P_1 and P_2 with data items *a, b,* and *c* and P_3 with data items *a, c*. Consider the following transactions T_1 and T_2 executed at P_1, concurrently.

$$T_1: w_1(a)w_1(c),\ T_2: r_2(c)w_2(c)$$

Assume that the LDBS of P_1 produced the following schedule:

$$S_1=w_1(a)w_1(c)r_2(c)w_2(c)$$

Suppose that peers P_2 and P_3 are connected with P_1. Hence, after execution of T_1 and T_2, P_1 propagates them to P_2 and P_3. Assume that when P_2 executes the transactions it also executes the following local transaction L_2.

$$L_2=r_{L2}(a)r_{L2}(c)$$

Assume that the LDBS of P_2 generates the following schedule.

$$S_2=r_{L2}(a)r_{L2}(c)w_1(a)w_1(c)r_2(c)w_2(c),$$

Similarly, when P_3 receives the transactions it also executes them using its LDBS and produces the following schedule:

$$S_3=w_1(a)r_2(c)w_2(c)w_1(c)$$

If we observe we find that the resulting serialization orders of T_1 and T_2 at P_1, P_2, and P_3 are as follows:

$$SO_1: T_1 \rightarrow T_2,$$
$$SO_2: L_2 \rightarrow T_1 \rightarrow T_2,\ SO_3: T_2 \rightarrow T_1$$

Notice that each local schedule in each peer is serializable, but the serialization order or conflict relationship of T_1 and T_2 in the schedule S_3 at P_3 is different with respect to the schedule S_1 at P_1.

For ensuring database consistency in acquainted

peers over the acquaintances with respect to the local execution of transactions in a peer, the serialization order of the transactions must be same in all the acquainted peers if there are direct conflicts between transactions. It turns out to be more difficult when local transactions cause indirect conflict between global transactions that may not conflict directly when they are initiated. Moreover, we need to consider several failure-prone situations during execution of transactions that can occur. For examples, a peer may go offline when a transaction is active in the network, a peer may fail due to power fails or the system crashes, or a peer has successfully executed a transaction but the transaction has failed in one of its acquaintees. Examples of a transaction failure are a transaction abort to timeout, or a failure to pass the validation test by the local transaction manager. We can consider an offline status of a peer as a failure of a peer. Ensuring consistent serialization order of transactions in all participating peers considering failure-prone situations is another important concern in a data sharing system.

1.2 Objectives and Assumptions

Objectives:

This paper proposed an approach for ensuring a consistent execution view of concurrently executing transactions in a data sharing system. A transaction is a sequence of read (e.g. SQL select) and write (e.g. SQL update, delete, and insert) actions in a database. Although we assume that each LDBS of each peer guarantees serializability, but the transactions that execute concurrently in multiple peers, may have different execution views at different peers. We first identify some potential problems for ensuring a consistent execution view of transactions in a data sharing system, and introduce correctness criteria in order to ensure the consistent execution view of transactions. We also propose one approach for ensuring this correctness condition, namely the Enforce Serialized Order method without violating the autonomy of local peer database management systems.

Assumptions:

1. When a user submits a transaction in a peer, he/she is only aware of the local database schema, and there is no global transaction manager or coordinator in the system.

2. A peer is not able to control or synchronize the execution of transactions in another peer.

3. Each LDBS has a mechanism for ensuring the local serializability.

2. Preliminaries

For ease of presentation, we use the well-known read-write model of transactions. We now recall the basics of this model. Let a database be a (finite) set $D=\{a,b,c,...\}$ of data objects. A transaction T is a sequence of database operations applied to a subset of data objects D. Formally, $T=(O_T,<_T)$, where O_T, is a finite set of operations and $<_T$ is a partial order of operations that have been invoked by a transaction T. The operations of a transaction T consists of *reads* (denoted by r(a)) and *writes* (denoted by w(a)) operations. Further, each T has begin and termination operations commit (c) or abort (a).

The concurrent execution of transactions results in a schedule. A schedule S is a pair $(\Gamma_S,<_S)$, where Γ_S is a finite set of transactions and $<_S$ is a partial order over the operations of transactions in Γ_S. The partial order $<_S$ satisfies the property that it preserves the order of steps within each transaction, (that is, $<_{T_i} \subseteq <_S$, for each $T_i \in \Gamma_S$.

The most commonly used correctness criteria for an acceptable schedule is conflict serializability [10]. Consider a schedule S consists of transactions T_i and T_j, then a transaction T_i is said to conflict (direct conflict) with T_j, denoted by $T_i \rightarrow T_j$, if there exist operations o_i in T_i and o_j in T_j, $T_i \neq T_j$, such that $o_i <_S o_j$, and o_i, o_j access the same data item and one of them is a write operation. By $\rightarrow^*$, we denote the transitive closure (indirect conflict) of the $\rightarrow$ relation.

In our work, we call the conflict relation between transactions as serialization order. The execution views of a set of transactions Γ in two schedules S_i and S_j are same if the serialization orders of the transactions are same in their execution. We assume the commit order of two transactions as the serialization order if there is no conflict between the transactions. A schedule S is called conflict serializable (serializable) if there exists a serial schedule S' such that the transactions in S have the same serialization order as in S'.

Similar to the execution semantics and classification of updates [20], transactions can be classified into three categories, namely *local*, *remote*, and *global*. We denote a transaction initiated in a peer P_i by T_i. If the transaction is local, that is the transaction is executed only in the local database at P_i, then we denote it by L_i. A remote transaction generated by a

peer P_i from a transaction T_i for its acquaintee P_j is denoted T_{ij}. A *global* transaction is denoted by G_i, which is a set of remote transactions. For ease of presentation, we denote remote transactions $T_{ij}, T_{ik}, \ldots$ generated from T_i as T_i, since they are actually generated from T_i, and a global transaction G_i is represented with the initiator T_i. Intuitively, execution of any component transaction $T_i, T_{ij}, T_{ik}, \ldots$ is called the execution of G_i.

2.1 Properties of a Global Transaction

In a data sharing system, a global transaction consists of a set of transactions that includes a global transaction initiator and a set of remote transactions. Each of the transaction is called a component transaction of the global transaction. Note that each component transaction is an atomic transaction resulted from the translation of another component transaction. Each component transaction accesses data items that are located in the peer where the transaction is active. Unlike a global transaction in an MDBS, a component transaction is not decomposed into subtransactions to access data at acquaintees. In order to access data at acquaintees, the component transaction is propagated as an atomic transaction after translation to each of the acquaintees if there are data mappings between the acquainted peers with respect to the data accessed by the transaction.

3. Problems for Maintaining Consistent Executions of Transactions

A classic technique for preserving database consistency during concurrent execution of transactions is to organize interleaving transactions such that their executions are atomic, recoverable, and serializable. However, classic serializability has known shortcomings when used as a correctness criterion for distributed computing environments, such as multidatabase systems, transactional workflow executions [11], or P2P systems. First, it requires close coordination and interaction among sites, that is, the sites must agree on the execution of global transactions in a specific and consistent manner. Second, as distributed transactions tend to be long lived, the use of serializability as a correctness criteria would restrict data availability [12].

One of the important issues for distributed multidatabase systems is to maintain global serializability of global transactions without violating the autonomy of local databases. The main problem occurs due to indirect conflicts between global transactions which cause different serial order of transactions at different sites. These problems have been widely studied and numerous solutions have been proposed, for example, in [13,14, 15,16].

Generally, in an MDBS, the global transaction manager (GTM) plays an important role to ensure the global serializability of global transactions in the system. However, in a data sharing system there is no GTM, and transactions are executed first locally in each peer before being forwarded to the acquaintees. Since global transactions are propagated in a data sharing system from peer to peer along the acquaintances, the globally consistent execution of concurrently executing global transactions in a data sharing system can be achieved by ensuring the consistent execution in each acquaintance that is included in the propagation paths of the global transactions. With respect to the execution of transactions in an acquaintance (i,j), the acquaintance level consistent execution of transactions between P_i and P_j is maintained if the following two conditions are satisfied [17]:

1. All the operations of a transaction must be executed in the same order in peers P_i and P_j of an acquaintance (i, j).
2. For any concurrent execution of global transactions, it is required to maintain the consistent execution of the transactions over all the acquaintances in the propagation paths of the global transactions. Formally, for any acquaintance (j,k) between P_j and P_k, if there are schedules $S_j=(\Gamma_{Sj},<_{Sj})$ and $S_k=(\Gamma_{Sk},<_{Sk})$ in P_j and P_k respectively, and each transaction in Γ_{Sk} is a translation of a transaction in Γ_{Sj}, then for all operations $o_1, o_2 \in S_k$, $o_1 <_{Sk} o_2$ iff $o_1 <_{Sj} o_2$

The first condition simply enforces the same execution order of the operations of a transaction at the peers in an acquaintance. The condition can be satisfied easily by forwarding each translated transaction as a single message to the acquaintees. Each acquaintee processes the transaction just like it processes its local transactions. Therefore, the order of the operations of a single transaction is maintained. In order to meet the second condition, we need to ensure the same execution views of transactions in each acquaintee of a peer. Note that the second condition cannot be fulfilled by sending the transactions serially according to the local serialization order of the sender since the sender has no knowledge about the execution order of the

transactions at a remote peer. In the following examples, we describe some of the problems that occur during the concurrent execution of global transactions over acquaintees. In Section 4, we present an approach to ensure consistent execution of global transactions over acquaintees.

Example 1[Direct conflict]
Consider a data sharing system shown in Figure 1. Assume that peers P_1 and P_2 have data items {a, b, c}, and P_3 has data items {a, c}. Suppose that the transactions T_1 and T_2 executed at P_1 concurrently, and P_1 produced the schedule S_1 as follows:

$$T_1{:}w_1(a)w_1(c),\ T_2{:}r_2(c_1)w_2(c_1)$$
$$S_1{=}w_1(a)w_1(c)r_2(c)w_2(c)$$

Based on the data accessed by T_1 and T_2, P_1 forwards them to peers P_2 and P_3. For ease of presentation, we keep the same notation of T_1 and T_2 in P_2 and P_3.

The forwarded transactions of T_1 and T_2 in P_2 and P_3 are as follows:

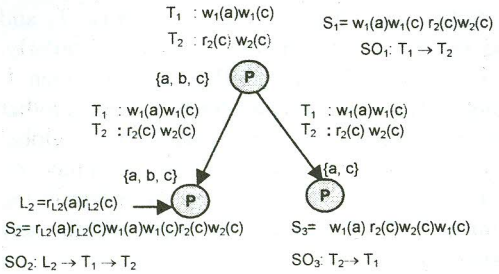


Figure 1: Inconsistent Serialization Order (Direct Conflict)

$$(P_2){:}T_1{=}w_1(a)w_1(c),\ T_2{=}r_2(c)w_2(c)$$
$$(P_3){:}\ T_1{=}w_1(a)w_1(c),\ T_2{=}r_2(c)w_2(c)$$

Assume that peer P_2 executes the following local transaction L_2 concurrently when it executes T_1 and T_2.

$$(P_2){:}\ L_2{=}r_{1,2}(a)r_{1,2}(c)$$

Consider that after receiving the transactions T_1 and T_2, P_2 and P_3 generated the following schedules.

$$S_2{=}r_{1,2}(a)r_{1,2}(c)w_1(a)w_1(c)r_2(c)w_2(c),$$
$$S_3{=}w_1(a)r_2(c)w_2(c)w_1(c)$$

Therefore, the resulting serialization orders of T_1 and T_2 at P_1, P_2, and P_3 are as follows:

$$SO_1{:}\ T_1{\rightarrow}T_2,\ SO_2{:}\ L_2{\rightarrow}T_1{\rightarrow}T_2,$$
$$SO_3{:}\ T_2{\rightarrow}T_1$$

Notice that each local schedule in each peer is serializable, but the execution view of T_1 and T_2 in the schedule S_3 at P_3 is different with respect to the schedule S_1 at P_1. Since each peer executes transactions independently, and there is no central controller, the resulting schedules at different peers may be different. In order to keep the peer databases consistent with each other, the execution view of the transactions should be the same in each peer.

Example 2[Indirect conflict]
In this example, we show how the local transactions cause different execution views of transactions in the acquaintees of a peer P_i even though the transactions have no conflict when the transactions executed at P_i. Consider Figure 2, where transactions T_1 and T_2 executed concurrently at P_1, and the local transaction manager at P_1 produced the schedule S_1:

$$T_1{:}w_1(a),\ T_2{:}\ w_2(b)w_2(c)$$
$$S_1{=}w_1(a)w_2(b)w_2(c)$$

Based on the data mappings, P_1 forwards the transactions to P_2 and P_3 as follows:

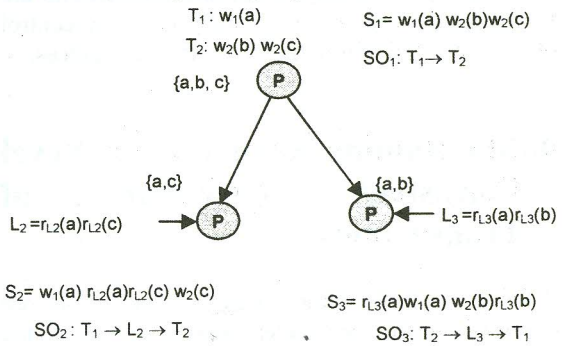


Figure 2: Inconsistent Serialization Order (Inirect Conflict)

$$(P_2){:}\ T_1{=}w_1(a),\ T_2{=}w_2(c),$$
$$(P_3){:}\ T_1{=}w_1(a),\ T_2{=}w_2(b)$$

Assume that the following local transactions executed at the same time when P_2 and P_3 received T_1 and T_2.

$$(P_2){:}\ L_2{=}r_{1,2}(a)r_{1,2}(c),$$
$$(P_3){:}\ L_3{=}r_{1,3}(a)r_{1,3}(b)$$

Consider that P_2 and P_3 generated the following schedules.

$$S_2{=}w_1(a)r_{1,2}(a)r_{1,2}(c)w_2(c),$$
$$S_3{=}r_{1,3}(a)w_1(a)w_2(b)r_{1,3}(b)$$

Notice that when T_1 and T_2 executed at P_1, there was no conflict between the transactions. Meanwhile, when T_1 and T_2 executed at P_3, they involved in

indirect conflict due to the presence of the local transaction L_3. Based on the execution views of T_1 and T_2 at P_2 and P_3, we observe the following serialization orders.

$$SO_2: T_1 \rightarrow L_2 \rightarrow T_2,$$
$$SO_3: T_2 \rightarrow L_3 \rightarrow T_1$$

Notice that the serialization orders of T_1 and T_2 at P_2 and P_3 are different. Hence, acquaintance-level consistent execution is not maintained.

In a multidatabase environment, the GTM has the control over the execution of global transactions and the operations they issue. The GTM can ensure the global serializability by a direct or indirect control of the global transactions. For example, altruistic locking [14], 2PC agent [18], site graph [19], and ticketing [13]. All the methods have a global transaction manager which plays an important role in ensuring the global serializability. Since in a data sharing system there is no such GTM, the only assumption we can make is that each peer ensures the local serializability. Once the transactions are forwarded to the acquaintees, the peer has no control of the execution of transactions at the acquaintees.

4. Maintaining Acquaintance-Level Consistent Execution of Transactions

Since in a data sharing system transactions are executed locally and independently in peers, the system does not require multi-site commit protocols (e.g. two-phase commit), which tend to introduce blocking and are thus not easily scalable. Specifically, in a peer data sharing system, transactions are executed locally, and then asynchronously propagated over acquaintances after their commitment. A consistent execution of transactions in a peer data sharing system can be achieved by ensuring the consistent execution of transactions in each peer over each acquaintance that is included in the propagation path of the transactions.

We observe from the examples in Section 3 that the inconsistent execution of transactions occurs at different peers due to the independent execution. However, for ensuring database consistency in acquainted peers over the acquaintances with respect to the local execution of a peer, the execution views of the transactions must be same in all the acquaintees of a peer if there are direct conflicts between transactions. Fortunately, we don't need to be worried about an indirect conflict between the transactions when it occurs in acquaintees. An indirect conflict occurs due to the local transactions in an acquaintee. When transactions have no conflict at the time they initiate in a peer, then these transactions can be executed in any order in the acquaintees. Since the data constraint property restricts [17] the access of the local and global transactions in a database, therefore, different execution views of transactions due to the indirect conflicts do not create the database inconsistency. This is because a global transaction does not read a data item that is written by a local transaction. The conflicts that can occur based on the data constraint property between a local transaction L and a global transaction T_1 are *write-read* and *read-write*. A write-read conflict between T_1 and L occurs for accessing a data item *a*, when a read operation of L is followed by a write operation of T_1. A read-write conflict occurs when a write operation of T_1 is followed by a read operation of L. Therefore, if L has a write-read conflict with T_1, then L does not create a read-write conflict for the same data item with another transaction T_2. This is because T_1 and T_2 had no conflict when they initiated. Similarly, when L has a read-write conflict with T_1, then L does not create a write-read conflict with another transaction T_2. Therefore, when two global transactions T_1 and T_2 execute at a peer and have no conflict, then their different execution orders in the acquaintees of the peer do not create any inconsistency.

Authors in [17] generalize the two problems, and introduce the notions for ensuring a consistent execution of transactions in a peer and its acquaintees. The notion is called acquaintance-level serializable.

The acquaintance-level serializability guarantees the same execution view of a set of transactions Γ in a peer and in its acquaintees. Although there is no GTM in a peer data sharing system, a globally consistent execution of transactions can be achieved by ensuring acquaintance-level serializability in each propagation paths of the transactions. In order to maintain the acquaintance-level serializability, we need to guarantee a consistent serialization order of the transactions at all the acquaintees of a peer. We know that when a set of transactions Γ is executed at a peer P_i, the transactions are executed immediately at P_i. Therefore, P_i generates a local schedule S_i without waiting for the execution of Γ in its acquaintees. After execution, P_i forwards Γ to its acquaintees. The execution and forward steps

continue until no propagation of Γ is possible. Each of the peer P_j executes Γ with the local concurrency control, and generates a local schedule S_j independently. The main challenge is how to guarantee a consistent serialization order in all S_j with respect to the order in S_i.

Authors in [17] proposed two approaches that guarantee acquaintance-level serializability. In this paper we propose another approach that also guarantee acquaintance-level serializability. In the following we describe the approach.

4.1 Enforced Serialization Order

We assume that each peer uses a function called *getSerializationGraph()* that returns the serialization graph (SG) of the currently executed global transactions. When a peer forwards the transactions to its acquaintees the peer also includes the SG with the transactions. When a peer receives the transactions, the LDBS executes the transactions under the control of local concurrency control and generates the serialization graph considering the edges of the SG that is received with the transactions. The SG logically creates a direct conflict between transactions. Therefore, if a peer executes the transactions in different order other than it is in the received SG, a cycle will be created. Thus, the peer will reject the execution of transactions.

In the following, we illustrate the method with an example.

Example 3 [Direct Conflict]
Consider *the example 1. The schedule generated from the execution of* T_1 *and* T_2 *is given below.*

$$S_1=w_1(a)w_1(c)r_2(c)w_2(c)$$

According to the method, the returnSerializationGraph() function returns the serialization graph SG_1: $T_1 \rightarrow T_2$ *at* P_1.

Based on mappings, P_1 *creates the following transactions for its acquaintees* P_2 *and* P_3 *and forwards them to* P_2 *and* P_3 *respectively including the serialization graph* SG_1 *with the transactions.*

$$(P_2): T_1=w_1(a)w_1(c),\ T_2=r_2(c)w_2(c)$$
$$(P_3): T_1=w_1(a)w_1(c),\ T_2=r_2(c)w_2(c)$$

Suppose P_2 *and* P_3 *receive the message* (T_1, T_2, $T_1 \rightarrow T_2$) *and assume that the following schedules and serialization graphs are generated by the respective LDBS.*

$$At\ P_2: S_2=r_{1,2}(a)r_{1,2}(c)w_1(a)w_1(c)r_2(c)w_2(c),$$
$$SG_2: L_2 \rightarrow T_1 \rightarrow T_2$$
$$\searrow T_1 \rightarrow T_2$$
$$At\ P_3: S_3=w_1(a)r_2(c)w_2(c)w_1(c)$$
$$SG_3: T_1 \rightarrow T_2 \rightarrow T_1$$

Note that the schedule S_3 *is not allowed by the local concurrency control of* P_3 *since a cycle is created. On the other hand, if the local schedule at* P_3 *were*

$S_3=w_1(a)w_1(c)r_2(c)w_2(c)$, *the corresponding serialization graph will be*

$$SG_3: T_1 \rightarrow T_2$$
$$\searrow T_2$$

which would be permitted by the local concurrency control at P_3 *and therefore ensures acquaintance-level serializability.*

Example 4 [Indirect Conflict]
Consider the example 2. The schedule generated from the execution of T_1 *and* T_2 *at* P_1 *is given at below.*

$$T_1: w_1(a),\ T_2: w_2(b)w_2(c)$$
$$S_1=w_1(a)w_2(b)w_2(c)$$

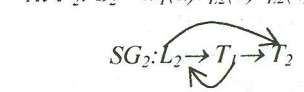
According to the method, the returnSerializationGraph() function returns the serialization graph SG_1: $T_1 \rightarrow T_2$ *at* P_1. *Assume here the execution order of the transactions is the serialization order.*

Based on mappings, P_1 *creates the following transactions for its acquaintees* P_2 *and* P_3 *and forwards them to* P_2 *and* P_3 *respectively including the serialization graph* SG_1 *with the transactions.*

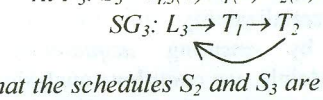
$$(P_2): T_1=w_1(a),\ T_2=w_2(c),$$
$$(P_3): T_1=w_1(a),\ T_2=w_2(b)$$

Suppose P_2 *and* P_3 *receive the message* (T_1, T_2, $T_1 \rightarrow T_2$) *and assume that the following schedules and serialization graphs are generated by the respective LDBS.*

$$At\ P_2: S_2= w_1(a)r_{1,2}(a)r_{1,2}(c)w_2(c),$$



$$At\ P_3: S_3= r_{1,3}(a)w_1(a)w_2(b)r_{1,3}(b)$$



Note that the schedules S_2 *and* S_3 *are not allowed by the local concurrency control of* P_2 *and* P_3 *since both schedules have cycles in graphs. On the other hand, if the local schedule at* P_2 *were*

$r_{1,2}(a)r_{1,2}(c)\ w_1(a)w_2(c)$ *or*
$w_1(a)w_2(c)r_{1,2}(a)r_{1,2}(c)$,

then the corresponding serialization graphs will be

$SG_2: L_2 \rightarrow T_1 \rightarrow T_2$ *or*

SG_2: $T_1 \rightarrow T_2$, L_2

which would be permitted by the local concurrency control at P_2 and therefore, ensures acquaintance-level serializability.

Similarly, if the local schedule at P_3 were
$w_1(a)w_2(b)r_{1,3}(a)r_{1,3}(b)$ *or*
$r_{1,3}(a)r_{1,3}(b)w_1(a)w_2(b)$

then the corresponding serialization graphs will be

SG_3: $T_1 \rightarrow T_2$, L_3 *or*

$SG_2: L_3 \rightarrow T_1 \rightarrow T_2$

which would be permitted by the local concurrency control at P_3 and therefore, ensures acquaintance-level serializability.

Our assumption of transaction processing in a peer data sharing system is that transactions are not generated continuously. We do not expect that users continuously submit update request in a P2P system. In a P2P system, generally, queries are more frequent than updates. Even, if transactions are continuous, according to our system, a peer forwards transactions to acquainted peers after the complete execution in the local peer.

5. Conclusion and Future Work

In this paper, we introduced a transaction model for a peer data sharing system. Our approach is scalable because a peer doesn't need any global knowledge of the system and there is no global coordinator. Transactions are processed by each peer independently and consistency is maintained recursively through acquaintances. A peer only ensures the serializability of its immediate acquaintances by ensuring *acquaintance-level serializability*. Mainly, we contribute analysis of the properties and semantics of transactions in a peer data sharing system, a correctness criterion for concurrent execution of transactions initiated at a single peer, and propose an approach ensuring global serializability without violating the autonomy of LDBSs.

A future goal is to investigate the transaction processing when global transactions initiated form many peers need to be executed concurrently in the system and analyze the correctness criteria for failure prone situation. Finally, we want to investigate these problems in a large peer network and show the scalability of the system.

References

[1] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management System. In *Proc. of the Int'l Conf. on Data Engineering*, pp 505-516, 2003.

[2] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer-Data Management System. In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(7):787-798, 2004.

[3] L. Serafini, F. Giunchiglia, J. Molopoulos, and P. Bernstein. Local Relational Model:A Logocal Formalization of Database Coordination. *Technical Report, Informatica e Telecomunicazioni, University of Trento*, 2003.

[4] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R.J. Miller, and J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. In *ACM SIGMOD Record*, 32(3):53-58, 2003.

[5] P. Rodriguez-Gianolli, M. Garzetti, L. Jiang, A. Kementsietsidis, I. Kiringa, M. Masud, R. Miller, and J. Mylopoulos. Data Sharing in the Hyperion Peer Database System. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pp. 1291-1294, 2005.

[6] A. Kementsietsidis, M. Arenas, and R.J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *Proc. of the Int'l Conf. on the Management of Data (ACMSIGMOD)*, pp. 325-336, 2003.

[8] W. Litwin. From Database Systems to Multidatabase Systems: Why and How. *British National Conference on Databases*, Cambridge Press, London, 1988.

[8] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. In *The International Journal on Very Large Data Bases*, 1(2):181-240, 1992.

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

[10] A. Zhang, M. Nodine, and B. Bhargava. Global Scheduling for Flexible Transactions in Heterogeneous Distributed Database Systems. In *IEEE Transactions on Knowledge and Data Engineering*, 13(3):439-450, 2001.

[11] M. Rusinkiewicz and A. Sheth. Specification and Execution of Transactional Workflows. In *Modern Database Management*, Addison-Wesley, 1995.

[12] H. Garcia-Molina and K. Salem. Sagas. In *Proc. of the Int'l Conf. on the Management of Data (ACMSIGMOD)*, pp. 249-259, 1987.

[13] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. In *IEEE Transactions on Knowledge and Data Engineering*, 6(1):166-180, 1994.

[14] 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):5-11, 1987.

[15] S. Mehrotra, R. Rastogi, Y. Breitbart, H.F. Korth, and A. Silberschatz. Overcoming Heterogeneity and Autonomy in Multidatabase Systems. In *Information and Computation*, 167(2):132-172, 2001.

[16] W. Du and A. Elmagarmid. Quasi Serializability: A Correctness Criterion for Global Concurrency Control in InterBase. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pp. 347-355, 1989.

[17] M. Masud and I. Kiringa. Acquaintance Based Consistency in an Instance-Mapped P2P Data Sharing System During Transaction Processing. In *Proc. of the Int'l Conf. on Cooperative Information Systems (CoopIS)*, pages 169-187, 2007.

[18] A. Wolski and J. Veijalainen. 2PC Agent Method: Achieving Serializability In Presence Of Failures in A Heterogeneous Multidatabase. In Proc. of Int'l Conf. on Databases, Parallel Architectures and Their Applications (PARBASE), pp. 321-330, 1990.

[19] Y. Breitbart and A. Silberschatz. Multidatabase Update Issues. In *Proc. of the Int'l Conf. on the Management of Data (ACMSIGMOD)*, pp. 135-142, 1988.

[20] MM Masud, I. Kiringa, H. Ural. Update Processing in Instance-Mapped P2P Data Sharing Systems. Int. J. Cooperative Information System 18(3-4): 339-379, 2009

Md. Mehedi Masud received his PhD in Computer Science at the University of Ottawa, Canada. He is an Assistant Professor at the Department of Computer Science, Taif University, KSA. His research interests include issues related to P2P and networked data management, query processing and optimization, and information security. He has published several research papers at national and international journals, conference proceedings.

Sultan Hamadi Aljahdali, Ph.D. secured B.S. from Winona State University, Winona, Minnesota in 1992, M.S. with honor from Minnesota State University, Mankato, Minnesota, 1996, and Ph.D. Information Technology from the Volgenau School of Information Technology and Engineering at George Mason University, Fairfax, Virginia, U.S.A, 2003. Currently Dr. Aljahdali is Dean of the college of computers and information systems at Taif University. His research interest includes software testing, developing software reliability models, soft computing for software engineering, computer security, reverse engineering, and medical imaging, furthermore he is a member of ACM, IEEE, and ISCA.