Maintaining Consistency in a Failure-Prone P2P Database Network During Transaction Processing

Md Mehedi Masud and Iluju Kiringa
SITE, University of Ottawa, 800 King Edward Road, Ottawa, ON, Canada
{mmasud,kiringa}@site.uottawa.ca

ABSTRACT

A peer database network consists of peers in which each peer has its own pre-existing local database system (LDBS). In a peer database network, the integration of sources is performed using pairwise mappings, and a peer shares its local database information with other peers by establishing data sharing constraints either by schema-level or data-level mappings. In our peer database network, we assume that each peer uses data-level mappings [7] for sharing data and resolving data heterogeneity. In this paper, we first introduce the execution semantics of a transaction in such a peer database network and formalize the problems of ensuring transaction execution consistency in the presence of failures of transactions. Since there is no global transaction manager and each peer executes transactions independently in a peer database network, different peers may have different execution views for the same set of transactions. Addressing this issue, we present an approach for ensuring a consistent execution of transactions in a peer database network.

1. INTRODUCTION

A peer database network (interchangeably called peer database system (PDBS)) combines both P2P and database management systems functionalities. In a peer database network, a peer provides access to its resources to other peers and shares its local data with other peers through the pairwise communication. In the last few years, steady progress has been made in research on various issues related to peer database systems, such as data integration models for peer databases [9], peer database mediation methods [8, 9], coordination mechanisms between peer databases [17, 18], and mapping data between peers [7, 8]. While the majority of peer data management systems focus mainly on providing facilities for query answering since it is one of the prime goals of P2P systems. With a few notable exceptions (discussed in the related work section), currently implemented P2P systems lack transaction management capabilities that are typically found in database management systems (DBMSs).

With respect to transaction processing, a peer database network is similar to a conventional multidatabase system (MDBS) [1, 4] in the sense that each system consists of a collection of independently created local database systems (LDBSs), and transaction management is handled at both the global and local levels. In an MDBS, a global level transaction is issued to the Global Transaction Manager (GTM), where the transaction is decomposed into a set of global subtransactions to be individually executed to corresponding LDBSs. Local transactions are directly submitted to the local database systems through a local application program interface. A local transaction manager executes both local transactions and global subtransactions autonomously. It is left to the GTM to ensure the atomicity of a global transaction that executes in different sites.

In contrast, a PDBS is built on a dynamic network of peers without a global transaction manager or controller. In a PDBS, a global level transaction is initiated by any peer and if the transaction needs to be executed in the network, then the transaction is propagated in the network through the pair-wise communication. Note that when a user submits a transaction to a peer, he/she is only aware of the local database schema and data vocabularies. The mappings between the peer, where the transaction is active, and other peers in the network determine the translation of the transaction and propagation and execution of the transaction to other peers. The logical connection, that is established through mappings, between two peers is called an *acquaintance*. The acquaintance is established either with data-level mappings [7] or schema-level mappings [8]. In this paper, we consider data-level mappings created from mapping tables [7] to establish an acquaintance between peers.

A classic technique for preserving database consistency 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 criteria for a distributed computing environments such as multi-databases, transactional workflow executions [22], 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 [5]. The authors in [19] proposes a correctness criteria for ensuring the transactions' execution consistency in peers over the acquaintances of a peer. However, the paper assumes that the transactions do not fail and all peers are online during the execution of transactions. Due to the dynamic nature, there are several situations may happen during the execution of transactions in peers over an acquaintance of a peer. For examples, a peer may become 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 acquainted peers. Examples of a transaction failure are transaction abort to timeout or failure to pass the validation test by the local transaction manger. In this paper, we are mainly concerned about the transaction failures and how these cause problems for ensuring the consistent execution of transactions in a peer and in its acquainted peers. In Section 4, we propose an approach for ensuring the consistent execution of transactions by taking into account only the failures of transactions.

1.1 Objectives, assumptions, and contributions

One of the objectives of this paper is to present an approach for ensuring a consistent execution of concurrently executing transactions in peers in a PDBS by taking into account the failures of transactions. In this paper, we assume that one single peer is initiating concurrent transactions. Although we assume that each LDBS of each peer guarantees serializability but transactions that execute concurrently in multiple peers may have different execution views in different peers due to failures of transactions.

In general, the model discussed in this paper is based on the following assumptions:

1. When a user submits a transaction in a peer, he/she is only aware of the local database schema and data vocabularies.

2. Transactions may fail in a peer during their execution.

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

4. Each LDBS has a mechanism for ensuring local serializability.

1.2 Contributions

Our contributions are as follows:

- We analyze the execution semantics of transactions initiated by a peer and identify the potential problems for ensuring a consistent execution of transactions when transactions fail in peers.

- We propose an approach for ensuring a consistent execution of transactions without violating the autonomy of LDBSs.

- Finally, we briefly discuss the transaction recovery mechanism in a PDBS.

2. PEER DATABASE SYSTEM MODEL

A PDBS is a set $P = \{P_1, P_2, \cdots, P_n\}$ of n peers with autonomous preexisting local database systems (LDBSs). Formally, each peer P_i $(1 \leq i \leq n)$ is defined by a pair of $< DB_i, M_i >$, where DB_i is a database and M_i is a set of mapping tables [7]. Each DB is a relational database with a finite universe of attributes $U = \{A_1, \cdots, A_l\}$. Each

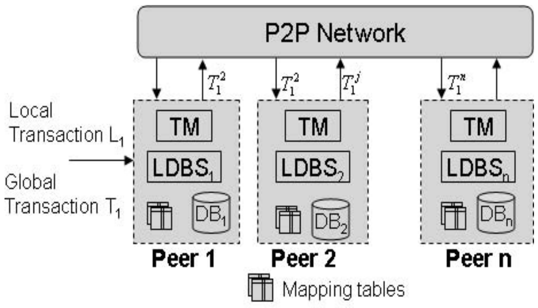


Figure 1: The P2PDBS model

attribute A_i is associated with a set of values called the domain of A_i and denoted by $dom(A_i)$. A non-empty subset of U is called a relation schema R. Each peer allows a subset of its attributes for data sharing. The peer is said to expose these attributes.

Consider two peers P_i and P_j that expose attributes U_i and U_j, respectively. A mapping table is a relation over the attributes X, Y, where $X \subseteq U_i$ and $Y \subseteq U_j$ are non-empty sets of attributes from the two peers. A tuple (a, b) in the mapping table indicates that the value $a \in dom(X)$ is associated with the value $b \in dom(Y)$. The recorded associations can be one-to-one, one-to-many or many-to-many. A set of mapping tables $M_{ij} = \{m_1, \cdots, m_k\} \subseteq M_i$ in P_i stores the data mappings between data items of P_i and P_j. The set M_{ij} allows P_i and P_j to share their local data. The construction of mapping tables M_{ij} forms an acquaintance (i, j) between P_i and P_j. Here, P_j and P_i are *acquaintees* of each other.

In a PDBS, when a user in a peer P_i submits a transaction T_i, he/she needs only be aware of the local database schema and data vocabulary. When a transaction T_i is submitted in P_i, the transaction is executed in P_i and appropriate actions are performed in DB_i. Whenever, a data is accessed by T_i stored in P_i, the data in each acquaintee P_j of P_i need to be accessed subject to the data mappings in M_{ij} for the data accessed by T_i. Therefore, when a transaction is submitted in a peer for execution, the peer looks for data mappings in mapping tables for the data accessed by the transaction. If there exist mappings for the data accessed by a transaction, then the transaction is translated using the mapping tables relevant to the transaction. A mapping table $m[X, Y]$ is relevant to T_i, if the data accessed by T_i has data mappings in m. After translating T_i, P_i forwards the translated transactions to the acquaintees which are relevant to T_i. An acquaintee P_j of P_i is *relevant* to a transaction T_i at P_i, if T_i can be translated for P_j over the acquaintance (i, j) with respect to the relevant mapping tables. When an acquaintee P_j of P_i receives a translated transaction from P_i, it also executes the transaction and translates the transaction for its acquaintees. Peer P_j also forwards the translated transactions to the relevant acquaintees. This *execute-and-forward* step is repeated in each of the relevant acquaintees, causing in turn a further propagation of the transaction in a PDBS. Figure 1 illustrates a sample PDBS model. The figure shows that peers are connected in a P2P network and each peer has a database attached to it along with a set of mapping tables. The figure also shows that a peer can initiate both local and global transactions; for example, peer 1 initiates a local transaction L_1 as well as a global transaction T_1. The global transaction is propagated throughout the network, for example to peer 2 after being translated as T_1^2

2.1 Transaction classifications

Based on the execution semantics, transactions are classified into three categories, namely *local*, *remote*, and *global* transactions.

-**Local transaction** (L_i): A transaction L_i which is submitted in a peer P_i and accessing only the local database DB_i.

-**Remote transaction** (T_i^j): From the point of view of a peer P_i, any local transaction T_i that is translated as T_i^j for an acquaintee P_j will result in a transaction that is called remote.

-**Global transaction** (G_i): A *global* transaction $G_i = \{T_i, T_i^j, T_i^k, \cdots\}$ consists of a transaction T_i (global transaction initiator) originated in P_i and a set of *remote* transactions $T_i^j (1 \leq j \leq n)$.

Note: For ease of presentation we denote remote transactions $T_i^j, T_i^k, \cdots$ as T_i since they are actually generated from T_i and a global transaction G_i is represented by T_i. Intuitively, execution of any component transaction $T_i, T_i^j, T_i^k, \cdots$ is called the execution of G_i.

2.2 Properties of a global transaction

In a PDBS, a global transaction consists of a set of transactions that includes a global transaction initiator and a set of remote transactions. Each of these 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 in a peer where the component 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 of a peer, a component transaction is propagated to each of the acquaintees as an atomic transaction after translation, if there are data mappings exist between the acquainted peers with respect to the data accessed by the transaction. In this paper, we do not describe the translation algorithm.

We now describe the structure of a global transaction G_i initiated from a transaction T_i in P_i and the relationship of component transactions of G_i. When P_i generates a set of remote transactions from T_i to be executed in its immediate acquaintees, G_i can be viewed as a two-level transaction. In this case, T_i becomes the root of G_i. G_i becomes a multi-level transaction when the acquaintees of P_i also generate remote transactions for their respective acquaintees. Consequently, a global transaction may have multiple layers depending on the number of hops it propagates. Intuitively, as transactions are propagated in the system acquaintance-by-acquaintance, a (directed) *transaction dependency graph* is induced. The nodes in this graph represent transactions and there is an edge from transaction T_i^j to transaction T_i^k, if T_i^k depends on T_i^j. A transaction T_i^k on peer P_k is said to depend on transaction T_i^j, if peer P_j and P_k are acquainted and transaction T_i^k has resulted from the translation and propagation of transaction T_i^j from peer P_j to P_k.

2.3 Data constraint property

Mapping tables store data sharing constraints by associating data vocabularies between peers. Although mapping tables impose constraints on the associations of values, they respect the autonomy of the sources whose values they associate. Based on constraints in mapping tables, we can categorize the data stored in a peer into two categories. The data in a peer that are shared using mapping tables are called shared data (SD) and the data that have no mappings in mapping tables are called local data (LD).

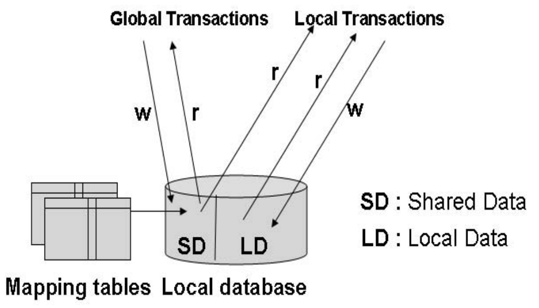


Figure 2: Data constraint property

In a peer database network, a transaction in a peer can access local as well as shared data items. Therefore, a peer is responsible to maintain the consistency of the local as well as shared data. An LDBS in a peer maintains the local database consistency when a transaction accesses only LD items. Meanwhile, if a transaction accesses SD items, then the consistency of SD must be maintained in the local peer as well as in acquaintees. Therefore, when an update occurs on an SD item x through the interface of LDBS in a peer, then the update must be propagated and executed in an acquaintee of the peer in order to maintain the mutual consistency on SD. We can classify the consistency on data items in a PDBS into two types:

Local consistency: the consistency of the local data items. The constraints are defined in the local database system.

Peer-to-Peer consistency: the consistency of the data items that are shared between peers. The constraints are defined using mapping tables when two peers share there data.

Therefore, the introduction of the peer-to-peer constraints, partitions the data items D_i in a peer P_i into local data (LD_i) and shared data (SD_i), such that $LD_i \cap SD_i = \phi$ and $D_i = LD_i \cup SD_i$. Intuitively, if there is an association of $a_i \in D_i$ and $a_j \in D_j$, $i \neq j$ in a mapping table, then data item a_i and a_j are shared data items in D_i and D_j, respectively. Figure 2 shows the categories of data. From the figure we notice that SD is mapped using the mapping tables, and LD is not mapped with any mapping tables. Note that mapping tables coexist with in a local database.

In order to avoid the inconsistency of shared data items between peers, the local transactions must be restricted to access data items in SD. In our work, we restrict a local transaction from updating a shared data item. A local transaction, however, is not restricted to read shared data items. Since mapping tables are placed between peers to share data, and global transactions are generated based on data mappings in mapping tables. Therefore, global transactions only access the shared data items, and they have full access on the shared data items. However, a global transaction is restricted to read and write local data items in a peer. This is logical since global transactions are used only to access shared data.

2.4 Preliminaries

For the purpose of this paper, 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 $D = \{a, b, c, \cdots\}$ of data objects. A transaction T is a sequence of database operations applied to a subset of data objects D. Formally, T=$(O_T, \prec_T)$, where O_T, is a finite set of operations and $\prec_T$ 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).

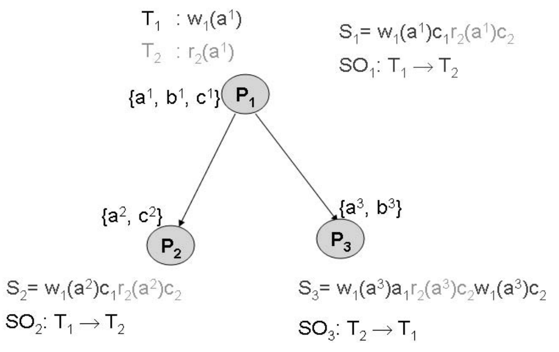
The concurrent execution of transactions results in a schedule. A schedule S is a pair $(\Gamma_S, \prec_S)$, where Γ_S is a finite set of transactions in S and $\prec_S$ is a partial order over the operations of transactions in Γ_S. The partial order $\prec_S$ satisfies the property that it preserves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

The most commonly used correctness criteria for a schedule is conflict serializability [11]. 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 $o_i \prec_S o_j$, and o_i, o_j access the same data item and one of them is a write operation. By $\overset{*}{\rightarrow}$ we denote the transitive closure (indirect conflict) of the $\rightarrow$ relation. In this paper, we call the conflict relation between transactions as serialization order. For a set of transactions Γ in two schedules S_i and S_j have the same execution views, if transactions in Γ have same serialization orders. When there is no conflict between transactions, we assume the commit order as the serialization order of the transactions. A schedule S is called conflict serializable (serializable), if there exists a serial schedule S' such that transactions in S have the same serialization order as in S'.

3. TRANSACTIONS SCHEDULING PROBLEMS DURING FAILURES OF TRANSACTIONS

In a PDBS, when a set of transactions $\Gamma = \{T_1, T_2, \cdots, T_n\}$ is executed concurrently in a peer P_i, the local database system generates a schedule S_i. The set Γ may be propagated to the acquaintees P_j $(1 \leq j \leq m)$. After receiving the transactions, each P_j independently generates its own schedule S_j. Since transactions are executed independently in peers, different peers may generate different execution views of transactions. The paper [19] proposes a mechanism for ensuring the same execution views of transactions over the acquaintances of a peer. However, the paper assumes that the transactions do not fail and all peers are online during the execution of transactions.

There are several failure-prone situations can occur in a PDBS. For examples, a peer may become 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 transaction abort to timeout or failure to pass the validation test by the local transaction manger. We can consider an offline status of a peer as a peer failure. In this paper, we are mainly concerned about the transaction failures and how these cause problems for ensuring the consistent execution views of transactions in a peer and in its acquaintees. If the transactions are not executed in the consistent order in each peer, the databases in peers may be inconsistent with respect to each other. The following examples show that a peer and its acquaintees may have different execution views for the same set of transactions when a transaction fails in a peer. We consider the situations of direct and indirect conflicts between transactions at the time transactions initiated in a peer. In Section 4, we propose an approach for ensuring the consistent execution view of transactions considering the failures of transactions.



T_1 : $w_1(a^1)$
T_2 : $r_2(a^1)$
S_1= $w_1(a^1)c_1r_2(a^1)c_2$
SO_1: $T_1 \rightarrow T_2$
$\{a^1, b^1, c^1\}$
P_1
$\{a^2, c^2\}$
$\{a^3, b^3\}$
P_2
P_3
S_2= $w_1(a^2)c_1r_2(a^2)c_2$
SO_2: $T_1 \rightarrow T_2$
S_3= $w_1(a^3)a_1r_2(a^3)c_2w_1(a^3)c_2$
SO_3: $T_2 \rightarrow T_1$

Figure 3: Direct conflict

Example 1 (Direct conflict). *Consider a PDBS with three peers shown in Figure 3. Assume that peer P_1 has data items $\{a^1, b^1, c^1\}$, P_2 has data items $\{a^2, c^2\}$, and P_3 has data items $\{a^3, b^3\}$. Suppose that transactions T_1 and T_2 executed in P_1 concurrently and the local database system in P_1 produced the schedule S_1:*

$$T_1 : w_1(a^1),\ T_2 : r_2(a^1)$$
$$S_1 = w_1(a^1)c_1r_2(a^1)c_2$$

Suppose that the following data mappings exist in the acquaintances.

$$(1,2)\text{: } a^1 \rightarrow a^2,\ c^1 \rightarrow c^2,\ \ (1,3)\text{: } a^1 \rightarrow a^3,\ b^1 \rightarrow b^3.$$

Based on the data accessed by T_1 and T_2, P_1 translates T_1 and T_2 and forwards them to P_2 and P_3. The translation of T_1 and T_2 for P_2 and P_3 are as follows:

$$(P_2)\text{:}T_1{=}w_1(a^2),\ T_2{=}r_2(a^2),$$
$$(P_3)\text{:}T_1{=}w_1(a^3),\ T_2{=}r_2(a^3)$$

For ease of presentation, we keep the same notation of T_1 and T_2 and their translations. Assume that after receiving the transactions, P_2 and P_3 generates the following schedules.

$$S_2 = w_1(a^2)c_2r_2(a^2)c_2,\ S_3 = w_1(a^3)a_1r_2(a^3)c_2w_1(a^3)c_1$$

The resulting serialization orders of transactions T_1 and T_2 in peers P_1, P_2, and P_3 are as follows:

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

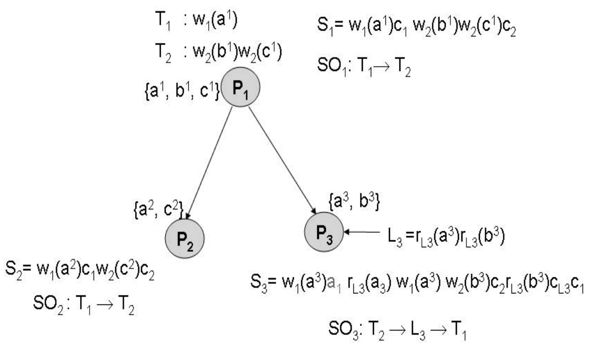


Figure 4: Indirect conflict

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 of P_3 is different with respect to the schedule S_1 in P_1. The execution view is different since T_1 has aborted in P_1 and later is resubmitted after the execution of T_2.

Example 2 (Indirect conflict). *In this example, we show how local transactions in a peer cause different execution views of transactions in acquaintees of a peer P_i even though transactions have no conflict when they executed in P_i. Consider Figure 4 where transactions T_1 and T_2 executed in P_1 and the local transaction manager in P_1 produced the schedule S_1:*

$$T_1 : w_1(a^1),\ T_2 : w_2(b^1)w_2(c^1)$$
$$S_1 = w_1(a^1)c_1w_2(b^1)w_2(c^1)c_2$$

According to the data mappings, P_1 generates the following transactions for P_2 and P_3 and forwards the translated transactions to them. Note that T_2 has been translated partially for P_2 and P_3 due to data mappings.

$$(P_2) : T_1{=}w_1(a^2),\ T_2{=}w_2(c^2),$$
$$(P_3) : T_1{=}w_1(a^3),\ T_2{=}w_2(b^3)$$

Assume that peer P_3 has a local transaction L_3 when P_3 executes T_1 and T_2:

$$(P_3) : L_3 = r_{L3}(a^3)r_{L3}(b^3)$$

Consider that P_2 and P_3 generates the following schedules.

$$S_2 = w_1(a^2)c_1w_2(c^2)c_2,$$
$$S_3 = w_1(a^3)a_1r_{L3}(a^3)w_1(a^3)w_2(b^3)c_2r_{L3}(b^3)c_{L3}c_1$$

Notice that when T_1 and T_2 executed in P_1, there was no conflict between the transactions. Meanwhile, when T_1 and T_2 are executed in P_3, they involve in indirect conflict due to the presence of a local transaction L_3 in P_3. Based on the execution views in P_2 and P_3, we observe the following serialization order.

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

Note that the serialization orders of T_1 and T_2 in P_2 and P_3 are different. In P_3, T_1 fails and later is resubmitted, this causes different execution view of transactions in P_3. Since in peers P_1 and P_2, T_1 and T_2 have no conflict, we consider commit order as serialization order.

In a failure-prone environment, a two phase commit (2PC) protocol [15] is needed to guarantee the atomic commitment of subtransactions of a multi-site transaction processing environment [20]. Since an LDBS of a site in such an environment may not support a visible prepared state of the 2PC protocol, therefore, a site uses 2PC agent or server to simulate the 2PC protocol between a GTM and 2PC agents [10] and servers. Meanwhile, in a PDBS there is no such GTM. The only assumption we can make that each peer ensures the local serializability and recoverability of transactions. Once a peer sends transactions to its acquaintees, the peer has no control of the execution of transactions in its acquaintees.

4. ACQUAINTANCE-LEVEL CONSISTENCY IN A PDBS

Since in a PDBS, a transaction is executed locally and independently, 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 PDBS, transactions are executed locally, and then asynchronously propagated over acquaintances after their commitment. Therefore, a consistent execution view of transactions in a PDBS can be achieved by recursively ensuring the consistent execution view of transactions in peers over each acquaintance that is included in the propagation paths of the transactions.

We observe from the examples in Section 3 that the inconsistent execution views of transactions occur due to failure of transactions in acquaintees. For ensuring database consistency in peers over acquaintances with respect to the local execution in a peer, the execution views of transactions must be same in all the acquaintees of a peer if the transactions have direct conflict. Fortunately, we don't need to be worried about an indirect conflict between transactions when they occur in acquaintees. An indirect conflict occurs due to 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 acquaintees. Since data constraint property restricts the access of local and global transactions in a database, therefore, different execution views of transactions due to indirect conflicts do not create database inconsistency. This is because a global transaction does not read a data item that is write by a local transaction. The conflicts that can occur based on 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 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 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 read-write conflict with T_1 then L does not create write-read conflict with another transaction T_2. Therefore, when two global transactions T_1 and T_2 executed in a peer and had no conflict between them, then their different execution order in acquaintees of the peer does not create any inconsistency. In the following, we generalize the two problems and introduce notions for ensuring a consistent execution of transactions in a peer and its acquaintees.

Definition 1 (Acquaintance-Level Schedule). *An acquaintance-level schedule $\mathbb{S}_i$= $S_i \cup (\bigcup_{j=1}^{m} S_j)$ with* *respect to a schedule S_i in P_i for a set of transactions Γ is the union of the schedule S_i and all the schedules S_j in P_j $(1 \leq j \leq m)$, where each P_j is an acquaintee of P_i.*

Definition 2 (Acquaintance-Level Consistent Schedule). *An acquaintance-level schedule $\mathbb{S}_i$ is called acquaintance-level consistent with respect to a schedule S_i in P_i for a set of transactions $\Gamma = \{T_1, T_2, \cdots, T_n\}$ and all schedules S_j in P_j over each acquaintance (i, j), $(1 \leq j \leq m)$ if*

1. *all the local schedules in $\mathbb{S}_i$ are serializable and*
2. *for any two transactions T_1 and T_2 in S_i, if there exist a serialization order (SO) $T_1 \rightarrow T_2$, then for all schedules $S_j \in \mathbb{S}_i (i \neq j)$, the SO is consistent between T_1 and T_2*

Theorem 1. *Consider a set of transactions $\Gamma = \{T_1, T_2, \cdots, T_n\}$ executed in a peer P_i and P_i generated a schedule S_i. The execution view of Γ is consistent over a propagation path $(P_i \rightarrow \cdots \rightarrow P_z)$ with respect to a schedule S_i in P_i, if for each acquaintance in $(P_i \rightarrow \cdots \rightarrow P_z)$, Γ is acquaintance-level consistent schedule.*

Although there is no GTM in a PDBS, but a consistent execution view of transactions can be achieved in a PDBS by ensuring *acquaintance-level* consistent view in each acquaintance in each propagation path of transactions. In order to ensure *acquaintance-level* consistent view, we need to guarantee the consistent order of the transactions in all the acquaintees of a peer. In the following, we propose a method that ensures *acquaintance-level* consistent view of transactions.

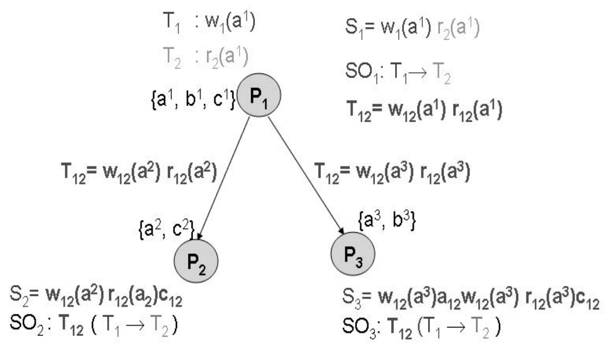
4.1 Maintaining acquaintance-level consistent execution of transactions

The *acquaintance-level* consistent execution ensures that each acquainted peer P_j of P_i relevant to Γ must generates the same execution view of Γ as it is generated by P_i. The consistency is maintained recursively from the initiator of Γ. In this method, we assume that a function called *return-Schedule()* is used by the initiator which returns a schedule of Γ. Note that the function returns only the operations in Γ in the order appeared in the schedule. We assume that the function is added externally in the system. This function can be easily developed by implementing an interface between the transaction's submission interface and the DBMS. A peer treats the schedule returned by the function as an atomic transaction, and translates the schedule for each of its acquaintee. The peer then forwards the schedule as a new transaction to its acquaintees. When an acquaintee receives the schedule (now transaction), the acquaintee processes the schedule as it processes a transaction. Note that treating a schedule as a transaction keeps the order of the operations of the original transactions since the order of the interleaved operations in the schedule remains same during translation.

Now we describe the method with examples. We show that the *acquaintance-level* consistent view is maintained considering both the direct and indirect conflict between transactions.

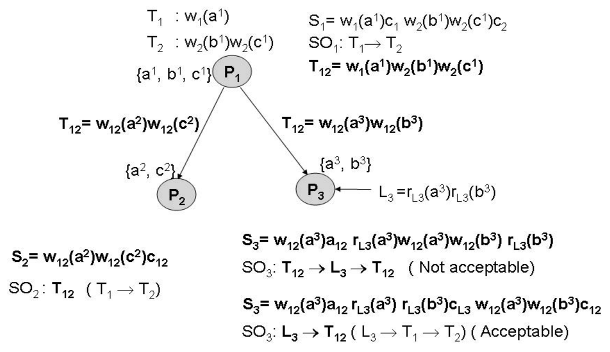
Example 3 (Direct Conflict). *Consider the situation of Example 1. The local schedule generated in P_1 is as follows:*

$$S_1 = w_1(a^1)c^1r_2(a^1)c^2.$$



T_1 : $w_1(a^1)$
T_2 : $r_2(a^1)$
$S_1 = w_1(a^1)\, r_2(a^1)$
SO_1: $T_1 \rightarrow T_2$
$T_{12} = w_{12}(a^1)\, r_{12}(a^1)$
$\{a^1, b^1, c^1\}$ P_1
$T_{12} = w_{12}(a^2)\, r_{12}(a^2)$
$T_{12} = w_{12}(a^3)\, r_{12}(a^3)$
$\{a^2, c^2\}$
$\{a^3, b^3\}$
P_2
P_3
$S_2 = w_{12}(a^2)\, r_{12}(a_2)c_{12}$
SO_2: T_{12} $(T_1 \rightarrow T_2)$
$S_3 = w_{12}(a^3)a_{12}w_{12}(a^3)\, r_{12}(a^3)c_{12}$
SO_3: T_{12} $(T_1 \rightarrow T_2)$

Figure 5: Maintaining acquaintance-level consistent execution during direct conflict



T_1 : $w_1(a^1)$
T_2 : $w_2(b^1)w_2(c^1)$
$S_1 = w_1(a^1)c_1\, w_2(b^1)w_2(c^1)c_2$
SO_1: $T_1 \rightarrow T_2$
$T_{12} = w_1(a^1)w_2(b^1)w_2(c^1)$
$\{a^1, b^1, c^1\}$ P_1
$T_{12} = w_{12}(a^2)w_{12}(c^2)$
$T_{12} = w_{12}(a^3)w_{12}(b^3)$
$\{a^2, c^2\}$
$\{a^3, b^3\}$
P_2
P_3
$L_3 = r_{L3}(a^3)r_{L3}(b^3)$
$S_2 = w_{12}(a^2)w_{12}(c^2)c_{12}$
SO_2: T_{12} $(T_1 \rightarrow T_2)$
$S_3 = w_{12}(a^3)a_{12}\, r_{L3}(a^3)w_{12}(a^3)w_{12}(b^3)\, r_{L3}(b^3)$
SO_3: $T_{12} \rightarrow L_3 \rightarrow T_{12}$ (Not acceptable)
$S_3 = w_{12}(a^3)a_{12}\, r_{L3}(a^3)\, r_{L3}(b^3)c_{L3}\, w_{12}(a^3)w_{12}(b^3)c_{12}$
SO_3: $L_3 \rightarrow T_{12}$ ($L_3 \rightarrow T_1 \rightarrow T_2$) (Acceptable)

Figure 6: Maintaining acquaintance-level consistent execution during indirect conflict

According to the method, P_1 creates a transaction T_{12} for its acquaintees P_2 and P_3 from the schedule S_1. The order of operations in T_{12} follows the order as mentioned in the schedule S_1. Figure 5 shows the scenario.

$$(P_2) : T_{12} = w_{12}(a^2)r_{12}(a^2),\ (P_3) : T_{12} = w_{12}(a^3)r_{12}(a^3)$$

Consider the same execution views of operations in P_2 and P_3 as in Example 1.

$$S_2 = w_{12}(a^2)r_{12}(a^2)c_{12},\ S_3 = w_{12}(a^3)a_{12}w_{12}(a^3)r_{12}(a^3)c_{12}$$

Note that in peer P_3, after the operation w_{12} the transaction T_{12} aborts. Therefore, no more operations of T_{12} will be executed until w_{12} successfully executes. After another try T_{12} is executed. Hence, the order of operations of T_1 and T_2 in P_3 remains same as it is in P_1. Therefore, $acquaintance - level$ consistent view is maintained in P_1, P_2, and P_3 with respect to the schedule S_1.

Example 4 (Indirect conflict). *Consider the Example 2, there is no conflict between T_1 and T_2 when they were executed in P_1. Assume that the following schedule is generated in P_1.*

$$S_1 = w_1(a^1)w_2(b^1)w_2(c^1)$$

According to the method, P_1 creates the following transactions for its acquaintees P_2 and P_3 from the schedule S_1. Figure 6 shows the scenario.

$$(P_2) : T_{12} = w_{12}(a^2)w_{12}(c^2),\ \ (P_3) : T_{12} = w_{12}(a^3)w_{12}(b^3)$$

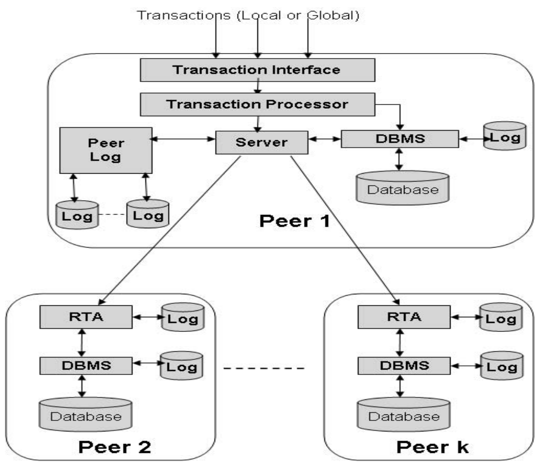


Figure 7: Transaction execution model

Assume that the following schedules result at P_2 and P_3 respectively. We consider the same execution sequence of operations as illustrated in Example 2.

$$S_2 = w_{12}(a^2)w_{12}(c^2)c_{12},$$
$$S_3 = w_{12}(a^3)a_{12}r_{L3}(a^3)w_{12}(a^3)w_{12}(b^3)r_{L3}(b^3)$$

Notice that the schedule S_3 is not allowed by the local concurrency control in P_3 since local schedule contains a cycle ($T_{12} \to L_3 \to T_{12}$). If the local schedule in P_3 is

$$S_3 = w_{12}(a^3)a_{12}r_{L3}(a^3)r_{L3}(b^3)c_{L3}w_{12}(a^3)w_{12}(b^3)c_{12}$$

then the schedule would be permitted by the local concurrency controller in P_3. Note that from the above execution, both P_2 and P_3 schedule the transactions T_1 and T_2 logically in the same order. Therefore, the *acquaintance-level* consistent view is maintained with respect to S_1.

4.2 Transaction execution model

In Figure 7, we briefly depict an execution model of transactions in a peer database network. Each peer provides a Transaction Interface (TI) which is used by the users to submit a transaction. When a transaction is submitted through TI, TI forwards the transaction to the Transaction Processor (TP). After receiving a transaction from TI, the Transaction Processor determines the execution scope of the transaction. If the execution scope of a transaction is local, then the transaction is directly submitted to the local database. If the scope is remote, then TP generates remote transactions for the acquaintees of the peer. TP then forwards the remote transactions to the server component. Server component then dispatches remote transactions to the corresponding Remote Transaction Agent (RTA) of each acquaintee. Since we give focus on the transaction execution aspect, we concentrate only on the functionality of RTA and server modules.

Server: A server in a peer P_i maintains a log table for each of its acquaintee which is called *peer log*, and monitors the execution status of a remote transaction that is propagated to an acquaintee. A peer log in P_i for an acquaintee P_j contains the status information of a transaction that is sent to P_j. When a peer P_i propagates a transaction T_i to a peer P_j, then P_i writes a log $< T_i, send >$ in the peer log of P_j. Peer P_i then waits for an acknowledgement for T_i. If P_i receives an acknowledgement for T_i, then P_i writes an entry $< T_i, received >$ in the peer log. If P_i does not receive an acknowledgement from P_j for T_i after a certain period of time, then P_i writes an entry $< T_i, failed >$ in the peer log. In this case, P_i assumes that P_j is offline or crashed. When P_j comes back online, P_i then tries again to send T_i. When T_i is successfully executed by P_j, P_j then sends a response to P_i. When P_i receives a response for T_i, P_i then writes an entry $< T_i, commit >$ in the log.

Remote Transaction Agent: This module is similar to a server module of a multidatabase system. When a RTA receives a remote transaction, it sends an acknowledgement to the server to inform about the reception of the transaction. Each RTA is responsible to submit the operations of a remote transaction to its LDBS. A RTA module does not participate to an atomic commitment protocol with a server for executing a remote transaction like in a multidatabase system. Since each remote transaction is an atomic transaction for a specific peer, therefore it does not require to wait for the execution of another remote transaction that is executing in another peer. To maintain the autonomy of LDBSs concurrency mechanism, each RTA is viewed by an LDBS as an application process. Therefore, a remote transaction is processed by an LDBS like a local transaction is processed. However, a RTA monitors the execution of operations of a transaction that is sent to the LDBS. For recovery purpose of a transaction, each RTA maintains a log table for each remote transaction it processes. A RTA uses the transaction recovery scheme [14] as used by a server in a multidatabase system. During execution of a transaction, a RTA generates a log record for each operation of a transaction. A transaction executing in an LDBS may have four different states. A transaction is said to be *active*, when no termination operation of the transaction has been submitted to the LDBS by the RTA. When RTA submits a commit operation, the transaction enters the *to-be-committed* state. If the commit operation submitted by the RTA has been successfully executed by the LDBS, the transaction enters the *committed* state. If the transaction aborts, it enters *aborted* state. Generally, an LDBS aborts a transaction when a transaction is involved in local deadlock or a transaction faces some internal error condition in the LDBS. After such aborts, the effect of failed transactions are undone by LDBSs and locks held by the aborted transaction are released. When a transaction is aborted, the RTA resubmits the transactions. If the failures occur after the transaction in *to-committed-state*, the write operations are resubmitted to the LDBS. When a remote transaction is successfully executed in a peer, the peer then sends the response to the sender of the remote transaction.

5. RELATED WORK

Until now there have been many algorithms proposed for transaction recovery scheme for MDBSs [6, 3, 2]. The solutions are not directly applicable in PDBSs due to the absence of a global coordinator in the systems and arbitrary topology of P2P networks. Recently, very few researchers attempted to focus on transaction processing in P2P networks. Now we describe some of the solutions.

In [21], authors present a preliminary proposal for peer-to-peer e-business transaction processing system. More specifically, the paper focuses on requirements analysis on different aspects of the collaboration and transaction procedure. However, it lacks precise semantics of transactions and does not describe the execution semantics of transactions.

In [16], a preliminary approach for agent-based transaction in a decentralized P2P network is presented. The focus is on a cooperative information system based on a P2P model. The model consists of a multi agent system with four components (wrapper, mediator, facilitator, and planner) which are responsible for the management and control of transactions composed by data management operations (read, write, delete) and their outcomes. However, the approach has no details.

In [13, 12], a concept of transaction processing in P2P environment is presented. The authors proposed a decentralized concurrency control for transactions relying on a decentralized serialization graph. Each peer and each transaction maintain a local serialization graph. The serialization graph of the peer reflects the dependencies of the transactions that invoked service calls on that peer whereas the serialization graph of the transaction includes the dependencies in which the transaction is involved. However, the strategy needs to modify the underlying database system to support the protocol. Also, it needs an application layer for creating the transactions agents, managing lock table, and processing the serialization graphs. In this paper, we assume that underlying system remains unchanged.

6. CONCLUDING REMARKS

In this paper, we present a transaction processing mechanism in a peer database system which consists of peers and each peer has its own local database system. We consider the transaction failures in the system and the mechanism for ensuring a consistent execution of transactions in peers. In the system, transactions are processed by each peer independently and consistency is maintained recursively along the acquaintances. A local peer only ensures the consistent execution of transactions in its immediate acquaintees by ensuring *acquaintance-level* consistent execution. We also propose an approach for ensuring the consistent execution of transactions. Finally, we describe the transaction processing and recovery mechanisms in the system. A future goal is to investigate the transaction processing when global transactions are initiated form many peers in the system and analyze the correctness criterion for such executions. Finally, we want to implement and investigate the mechanism in a large P2P network and show the scalability of the system.

7. REFERENCES

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

[2] S. Mehrotra, R. Rastogi, Y. Breitbart, H.F. Korth, and A. Silberschatz. Overcoming Heterogeneity and Autonomy in Multidatabase Systems In *Information and Computation*, Vol. 167, no. 2, 2001.

[3] Y. Breitbart, A. Silberschatz, and G. R. Thompson. Transaction Management Issues in a Failure-Prone Multidatabase System Environment. In *VLDB Journal*, Vol. 1, no. 1, 1992.

[4] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz Overview of Multidatabase Transaction Management. In *VLDB Journal*, Vol. 1, no. 2, 1992.

[5] H. Garcia-Molina and K. Salem. Sagas. In *ACM SIGMOD*, 1987.

[6] R. Alonso, H. Garcia-Molina, and K. Salem. Concurrency Control and Recovery for Global Procedures in Federated Database Systems. In *IEEE Data Engineering Bulletin*, Vol. 10, no. 3, 1987.

[7] A. Kementsietsidis, M. Arenas, and R.J. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *SIGMOD*, 2003.

[8] 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*, Vol. 16, no. 7, 2004.

[9] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management system. In *ICDE*, 2003.

[10] A. Wolski and J. Veijalainen. 2PC Agent Method: Achieving Serializability In Presence Of Failures In A Heterogeneous Multidatabase. In *PARBASE*, 1990.

[11] A. Zhang, M. Nodine, B. Bhargava. Global Scheduling for Flexible Transactions in Heterogeneous Distributed Database Systems In *IEEE Transactions on Knowledge and Data Engineering* , Vol. 13, no. 3, 2001.

[12] C. Schuler, H. Schuldt, C. Türker, R. Weber, and H. Schek. Peer-to-Peer Execution of (Transactional) Processes. In *International Journal of Cooperative Information Systems (IJCIS)*, vol. 14, no. 4, 2005.

[13] K. Haller, H. Schuldt, and C. Türker. Decentralized Coordination of Transactional Processes in Peer-to-Peer Environments. In *CIKM*, 2005.

[14] A. Brayner and T. Harder. Recovery in Multidatabase Systems. In *SBBD*, 1999.

[15] P. A. Bernstein, V. Hadzilacos, N. Goodman. Concurrency Control and Recovery in Database Systems. Addision-Welsey, 1987.

[16] L. Penserini, M. Panti, and L. Spalazzi. Agent-Based Transactions into Decentralised P2P. In *AAMAS*, 2002.

[17] 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.

[18] M. Arenas and V. Kantere and A. Kementsietsidis and I. Kiringa and R.J. Miller and J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. In *SIGMOD RECORD*, 2003.

[19] M. Masud and I. Kiringa. Acquaintance Based Consistency in an Instance-Mapped P2P Data Sharing System During Transaction Processing. In *CoopIS*, 2007

[20] San. Hwang, J. Srivastava, and J. Li Transaction Recovery in Federated Database Systems In , 1993

[21] S. Androutsellis-Theotokis, D. Spinellis, and V. Karakoidas. Performing peer-to-peer e-business transactions: A requirements analysis and preliminary design proposal. In *IADIS*, 2004.

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