Atomicity of Global Transactions in Multidatabase Systems

Yousef J. Al-Houmaily
Institute of Public Administration
Riyadh 11141, Saudi Arabia
Email: app0111@ipa.edu.sa

Abstract

We survey and classify the approaches proposed in the literature that facilitate the atomicity of global transactions in multidatabase systems. Our taxonomy is based on the assumptions made in the different approaches about the use of atomic commit protocols with externalized commit operators by the constituent database sites.

1 Introduction

A *multidatabase system* (MDBS) is a special type of distributed database systems that interoperates multiple, pre-existing and, probably, heterogeneous database systems. An MDBS allows each database system to continue to operate in an independent fashion and (ideally) does not require any changes to existing databases, applications and local database management systems (LDBMSs).

As shown in Figure 1, an MDBS consists of two software system components: (1) a *global transaction manager* (GTM) and (2) a set of *agents*. The global transaction manager is responsible for the management of global transactions that facilitate interoperability across the different database sites; whereas each agent is responsible for the different aspects of the execution of global transactions at its site and in particular, the atomic termination of global transactions and their recovery.

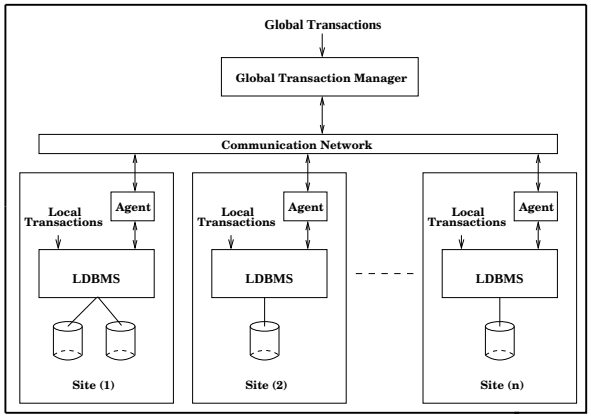

Figure 1. The MDBS Model.

Also shown in Figure 1 two types of transactions. These two types are:

- *local transactions* that access data located at only a single database. This type of transactions executes under the control of LDBMSs and the MDBS is not aware of their existence.
- *global transactions* that access data located at multiple databases under the control of the MDBS.

As in the case of (homogeneous) distributed database systems, a global transaction is decomposed by the GTM into several *subtransactions*, each of which executes as a local transaction at some site. To ensure the atomicity of a global transaction, the agents

at the sites where the transaction has executed cooperate with the GTM by engaging in the execution of an atomic commit protocol.

This paper deals with the atomicity issue in multidatabase systems that ensures the "all or nothing" principle of global transactions. Specifically, it contains a brief survey of the approaches that have been proposed in the literature and a taxonomy that relates these approaches to each other.

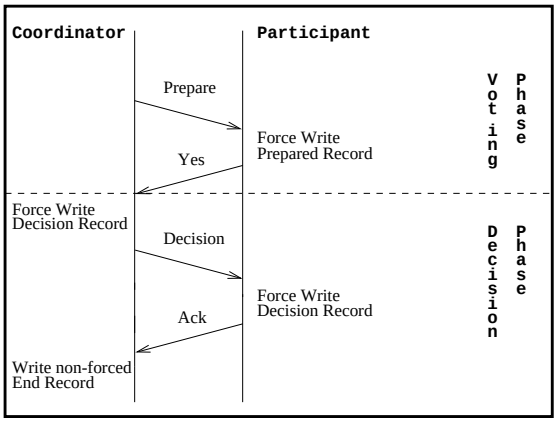

Figure 2. The basic two-phase commit protocol.

2 Atomicity in MDBSs

In any distributed database system, an *atomic commit protocol* (ACP) is needed to ensure the atomicity property of transactions. The *two-phase commit* protocol (2PC) [15, 18] is the simplest and most widely used ACP.

As shown in Figure 2, 2PC, as the name implies, consists of two phases, namely a *voting phase* and a *decision phase*. During the voting phase, the coordinator of a distributed transaction requests all the sites participating in the transaction's execution to *prepare to commit* whereas, during the decision phase, the coordinator either decides to commit the transaction if *all* the participants are *prepared to commit* (voted "yes"), or to abort if any participant has decided to abort (voted "no"). If a participant has voted "yes", it can neither commit nor abort the transaction until it receives the final decision from the coordinator. When a participant receives the final decision, it complies, *acknowledges* the decision and releases all the resources held by the transaction (i.e., releases the locks held by the transaction, removing the transaction control block from its table, etc.). The coordinator completes the protocol when it receives acknowledgments from all the participants.

The resilience of 2PC to system and communication failures is achieved by recording the progress of the protocol in the logs of the coordinator and the participants. The coordinator force writes a *decision* record prior to sending its final decision to the participants. Since a force write ensures that a log record is written into a stable storage that survives system failures, the final decision is not lost if the coordinator fails[1]. Similarly, each participant force writes a *prepared* record before sending its "yes" vote and a *decision* record before acknowledging a final decision. When the coordinator completes the protocol, it writes a non-forced *end* record, indicating that the log records pertaining to the transaction can be garbage collected when necessary. The end log record indicates to the garbage collection procedure that it can garbage collect all the log records pertaining to the transaction from the stable log.

Unlike (homogeneous) distributed database systems, the constituent LDBMSs in an MDBS might use different atomic commit protocols such as the *basic two-phase commit* protocol, that we discussed above, or one of its variants (see [1] for a survey of the most commonly known 2PC variants). Furthermore, some LDBMSs might not support any form of ACPs. For this reason, a LDBMS can be classified as either *externalized* or *non-externalized* LDBMS. An exter-

[1] In contrast, a non-forced log write is written into the log buffer in main memory and its cost is negligible compared to a forced write that requires a disk access. However, a non-forced log write might be lost in the case of a site failure.

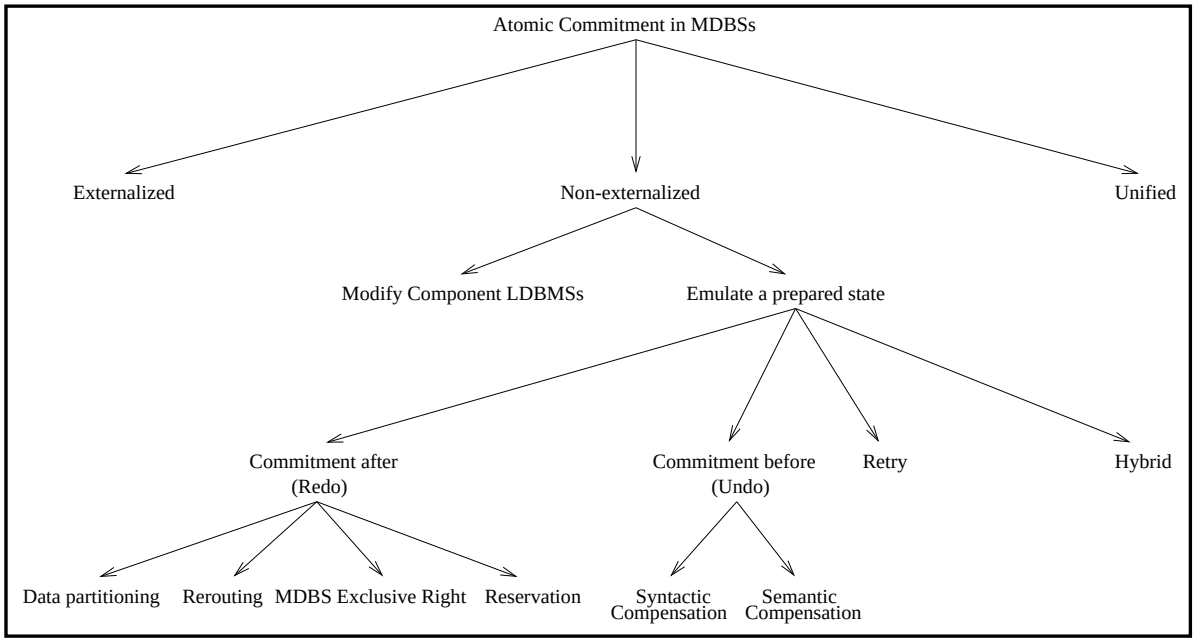

Figure 3. Taxonomy of atomic commitment in MDBSs.

nalized LDBMS: (1) implements an ACP and (2) makes the system calls pertaining to its commit protocol, called *commit operators*, available to the outside word through its interface. Otherwise, a LDBMS is called non-externalized.

Figure 3 depicts three approaches that ensure the atomicity of global transactions in a form of a taxonomy. The taxonomy is based on the two categories of LDBMSs that we discussed above. In the following sections, we discuss the motivation behind the research work in each of the three approaches and some of the methods that have been proposed to characterize them.

3 Externalized Approach

Given the current standardization efforts [4, 34], the research in this direction is based on the assumption that future LDBMSs will support ACPs with externalized commit operators. Thus, the challenge is to integrate LDBMSs that use different and incompatible ACPs. The incompatibility of ACPs means that the semantics of the coordination messages and the actions that are taken by a LDBMS that uses one ACP might be completely different than their counterparts in another ACP. Integrating LDBMSs that use incompatible ACPs in a MDBS is not a trivial task as it was previously believed [5, 29, 32, 33]. That is, it is not simply the case that once a LDBMS supports an externalized ACP, it can be integrated in an MDBS regardless of the ACPs used by the other LDBMSs [14, 30].

Some researches have concentrated in resolving the incompatibility of ACPs with respect to the semantics of the coordination messages. Hence, this group of researchers were interested in achieving *functional* correctness. That is, only achieving atomicity of global transactions. Other researchers looked at the atomicity of global transactions from a more pragmatic point of view. That is, achieving *operational* correctness in which the outcome of terminated transactions should, eventually, be able to be forgotten without sacrificing consistency. In the following two subsections, we overview the research in both directions.

3.1 Functional Correctness

Pu *et al* [29] concentrated on integrating *asymmetric* and *symmetric* classes of ACPs

in the Harmony prototype. Harmony integrates *centralized* LDBMSs where each externalizes one of the two classes of ACPs. In asymmetric protocols, such as 2PC, only the coordinator is responsible about making the final decision about the outcome of a transaction whereas in symmetric protocols, such as decentralized 2PC [30], all participating sites have the same right in the execution of the protocol and can commit the transaction independently.

In the Harmony project, a component system called *Supernova* is responsible for the translation of the semantics of the coordination messages of the protocols used by the different LDBMSs. In the event that some LDBMSs use asymmetric commit protocols while the others employ symmetric ones in a transaction's execution, Supernova is the coordinator of all asymmetric ACPs and a member of the symmetric ones. To ensure the atomicity of a transaction, Supernova does not send out its vote to the symmetric members until it hears from all LDBMSs that use asymmetric ACPs. Thus, the symmetric participants are prohibited from making a unilateral decision that might jeopardize atomicity until they receive the vote of Supernova.

Tal and Alonso [32, 33] took the problem one step further. In their work, they assume that each LDBMS is a *distributed* database system that externalizes an ACP. In this case, ensuring the atomicity of transactions is further complicated because a LDBMS, being distributed, might reach a different decision about the outcome of a transaction than the other participating LDBMSs in the transaction's execution. For example, if the GTM sends out prepare to commit messages to the LDBMSs participating in a global transaction's execution, the LDBMSs might reach different decisions about the outcome of the transaction if each one of them uses decentralized 2PC. This is because a prepare to commit message in decentralized 2PC has a dual role (i.e., it tells each LDBMS that the transaction has finished its execution and at the same time the GTM's vote). Thus, one LDBMS might commit the transaction if all the participants at its site have exchanged "yes" votes while another LDBMS might abort the same transaction if any participant at its site has decided to abort (i.e., voted "no").

By using *auxiliary* participant processes at each LDBMS, Tal and Alonso interoperate the basic 2PC, linear 2PC, decentralized 2PC and three-phase commit protocols. For example, in the case of a decentralized 2PC LDBMS, the auxiliary participant, which can be thought of as the agent in the MDBS model, is used to prohibit the other participants at its site from making a decision by not sending its vote to the participants at its site until it receives the final decision from the GTM. In this way, the participants are blocked from being able to make a commit decision that might latter jeopardize the atomicity of the transaction. Once the auxiliary participant receives the GTM decision, it propagates the decision to the other participants and completes the protocol.

3.2 Operational Correctness

The above two research works considered only functional correctness. That is, reaching a consistent decision regarding the outcome of global transactions. In this section, we overview the research that has considered operational correctness.

Mohan *et al* [22] proposed a new 2PC variant called the *generalized presumed abort* protocol (GPA). GPA is a modified version of the *presumed abort* protocol (PrA). The motivation behind this protocol is to achieve the best of the two worlds namely, the generality of the *peer-to-peer* and the performance of the *client-server* architectures, when considering ACPs. To allow for smooth migration from IBM's PrN, which is a 2PC variant that is adopted in the IBM SNA LU6.2 architecture, to IBM's GPA, the GPA protocol had to be a superset of the IBM's PrN protocol. Furthermore, the designers of GPA have concentrated on the implementation aspects of PrA

while taking into considerations some of the issues that are specific to the SNA LU6.2 architecture, such as supporting *heuristic decisions*. Hence, PrA had to modified in order to fit into SNA's architecture to achieve operational correctness.

Al-Houmaily *et al* [1, 2] explicitly defined operational correctness and showed the difficulties in achieving it even when interoperating the simplest ACPs. They used 2PC and its most commonly known two variants, namely presumed abort and presumed commit protocols, to demonstrate these difficulties. They also proposed the *presumed any* protocol that successfully interoperates the three 2PC variants according to operational correctness without modifying any of them.

4 Non-Externalized Approach

In the previous section, we discussed the methods that interoperates externalized LDBMSs. However, most of the literature in MDBSs consider legacy systems and is based on the assumption that each LDBMS does not externalize its ACP. Thus, the challenge is to ensure the atomicity of global transactions despite the fact that each LDBMS does not externalize an ACP. The approaches reported in the literature can be classified into two categories, as shown in Figure 3. The first category of protocols suggest modifying component LDBMSs to support an externalized ACP while the second category of protocols achieve the atomicity of global transactions by emulating a *prepared to commit* state. In this section, we overview the methods that characterizes both categories.

4.1 Modify Component LDBMSs

The researchers in the area of atomic commitment in MDBSs have realized the difficulties of ensuring the atomicity of global transactions and some have proposed the modification of component LDBMSs to support an externalized ACP [14, 29]. These proposals do not only violate the autonomy requirement of LDBMSs; but might be impossible to achieve. This is because even if the man power required to make such modifications is available, the source code of LDBMSs is usually copyrighted and might not be available for such modifications. Hence, this solution is, generally, not practical.

4.2 Prepared State Emulation

The other alternative that ensures the atomicity of global transactions is to emulate a *prepared to commit* state at each LDBMS site without having to modify the source code of the LDBMS. The protocols proposed in this direction still violate the autonomy of the database sites with variant degrees, as shown by Chrysanthis and Ramamritham [9], but less than the one that suggests modifying the source code of the component LDBMSs discussed above.

Emulating a prepared to commit state can be achieved using one of four approaches. The first two approaches are classified based on the relative commitment of the subtransactions pertaining to a global transaction at the participating LDBMSs with respect to the commitment the global transaction by the GTM [25]. That is, in the first approach called *commitment after*, the LDBMSs participating in a global transaction's execution commit the transaction locally after the GTM has made the final commit decision. Thus, in the event that a LDBMS aborts the transaction after the final commit decision is made but before it has received the decision, the effects of the transaction should be *redoable*. In the second approach called *commitment before*, LDBMSs commit a transaction before the GTM has made its final commit decision. Thus, in the event that the GTM has finally decided to abort the transaction, the effects of the transaction should be *undoable*. In the third approach, the effects of a global transaction are always *retriable*. That is, if a LDBMS aborts a global transaction locally, the GTM should be able to re-execute the transaction at the LDBMS repeatedly un-

til the transaction is finally committed at the LDBMS. The fourth approach called *hybrid* because it is basically a combination of the other three approaches.

4.2.1 Commitment After Approach

In this approach, the prepared to commit state is emulated by characterizing the *denied local updates* (DLU) concept [35, 36]. When a subtransaction enters the emulated prepared to commit state, the DLU requires that no local transaction should be able to modify the states of the data objects accessed by the subtransaction. In this way, if the LDBMS at the site where the subtransaction is in its prepared to commit state has decided to abort the transaction unilaterally for any reason, the subtransaction can be redone and produce the same effects that it has produced just before it has been aborted by resubmitting it again.

In the literature, there are four ways where the DLU concept is characterized. The first one is based on data partitioning. In this method, the type of a transaction determines which type of data the transaction can read and which type of data it can modify [6, 7, 8, 20]. For example, *globally updateable* data items can be only modified by global transactions whereas *locally updateable* data items can be only modified by local transactions. In the event that a LDBMS decides to abort a subtransaction and the final decision of the GTM is to commit the global transaction, the write operations of the subtransaction are redone by resubmitting a redo transaction that contains only the write operations of the subtransaction.

The second way is based on re-routing local transactions such that they have to go through the global transaction manager (or the agent at the site where they have been initiated) [25, 31]. Thus, the MDBS has complete control over local transactions and can abort any of them if DLU is violated.

The third way is the *MDBS exclusive right* [3, 11, 12, 36]. In the MDBS exclusive right, no local transaction is allowed to start executing at a LDBMS after a site failure until the MDBSs has recovered all transactions that were in their emulated prepared to commit states. After a failure, the agent of the MDBS is guaranteed exclusive access by the LDBMS administrators until it recovers all global transactions, denying any local transaction request to access the database system.

The fourth way is *reservations* [23, 24]. Reservations have two flavors. The first one called *generic* which is general enough to be used irrespective of the semantics of transactions and data while the other one is *semantics-based*. Both types of reservations characterize DLU by associating each pre-existing data item at each database site with a new one and modifying local transactions. In reservations, each subtransaction of a global transaction is associated with a reservation transaction. The reservation transactions of a global transaction are executed first. Only when all reservation transactions commits, the corresponding global transaction is executed. Otherwise, an unreservation transaction is executed for each committed reservation transaction.

In generic reservations, the extra data items serve as a form of high level locking to ensure the recoverability of global transactions by forcing them to conflict with local ones. This method is similar to the *ticket* method [13] that has been proposed to ensure serializability in MDBSs. When a reservation transaction executes, it marks each extra data item that is associated with the data item that will be accessed by the global transaction as “blocked”. If a reservation transaction encounters a “blocked” data item, it aborts and an unreservation transaction is executed for each successfully committed reservation transaction to “un-block” the data items that have been blocked by the corresponding reservation transaction. Local transactions are modified such that they abort if they attempt to modify a reserved data item (i.e., by checking its associated extra data item with respect to blocking). In this way, when the reservation subtransactions pertaining to a global transaction com-

mits, the global transaction is guaranteed to commit once it starts its execution and despite failures. If a failure occurs, the failed subtransactions are redone and their effects will be the same as if no failure has occurred because the DLU is preserved. Once a global transaction is committed, an unreservation transaction is executed to "unblock" the data items that the transaction has accessed, allowing other local transactions and reservations to access these data items.

The semantics-based reservations exploit the semantics of transactions and data and are similar in principle to the *escrow* method proposed by O'Neil [27] to enhance concurrency among transactions. However, semantics-based reservations have been proposed to ensure the atomicity of global transactions. In semantics reservations, a reservation transaction modify the extra data items such that no local transaction can cause the global transaction to abort after its associated reservation has committed. Notice that, unlike generic reservations, local transactions are allowed to access reserved data items. For example, if a subtransaction needs to reserve five seats on a plane, its associated reservation subtransaction will read the "available number of seats" data item to check the availability of the requested number of seats. Then, the reservation transaction will set the extra data item to five if these seats are available. Local transactions, in this example, are modified such that they abort if they attempt to modify the "available number of seats" data item below the value recorded in the extra data item.

As we mentioned above, semantics-based reservations are similar in principle to escrow. Unlike escrow, however, transactions in both types of reservations execute on original data items but not the extra (or reserved) items which is the case in the escrow method.

4.2.2 Commitment Before Approach

The atomicity of global transactions in the *commitment before* approach can be achieved in either a *syntactic* or *semantic* base. In the syntactic-based approach, the state of the database after the effects of a transaction has been rolled back is exactly the same as if the transaction has never modified any data items, which is the traditional notion of atomicity. On the other hand, the semantics-based approach uses a weaker notion of atomicity called *semantics atomicity* in which the state of the data objects after a transaction has been rolled back does not necessarily correspond to their states as they were before the transaction has modified them.

Consider, for example, an airline reservation application. Using the traditional notion of atomicity, when a reservation subtransaction aborts, the "available number of seats" data item associated with the plane that the transaction was making a reservation on, is exactly the same after the subtransaction has aborted as it was before the subtransaction has started. In semantics-atomicity, a reservation subtransaction is allowed to commit and if the whole global transaction is to be finally aborted, a *compensating* subtransaction is executed. Since other transactions might execute between the commitment of the subtransaction and its associated compensating subtransaction, the "available number of seats" data item might not be the same before the subtransaction has started.

To ensure syntactic compensation, no transaction should be able to access the data items modified by a transaction that is in its emulated prepared to commit state. This constraint can be achieved through re-routing of local transactions such that they have to access LDBMSs through the GTM (or the agents) [25, 28], as in the analogous method in commitment after approach. In this way, the GTM has complete control over local transactions and can prohibit any of them from accessing data modified by a prepared to commit transaction. Furthermore, the GTM has to be able to execute an inverse operation for each of the operations of an aborted transaction such that the states of the data items modified by a transaction that has been locally committed but finally aborted by the GTM is exactly the same after executing

the inverse operations. Thus, the GTM has to log the before images of the data items accessed by a prepared to commit global transaction. In this way, undoing the effects of an aborted global transaction is simply achieved by installing the before images of the data items that the transaction has modified [28].

On the other hand, semantic compensation can be achieved in two ways. In the first way, the subtransactions of a global transaction are *optimistically* committed in an individual manner. If a subtransaction is aborted, the GTM aborts the whole transaction. Therefore, a log is maintained in order to execute compensating transactions for the committed subtransactions. To ensure that a compensating transaction will eventually commit, no local transaction is allowed to execute between a subtransaction and its inverse compensating transaction. This is achieved by re-routing local transaction through the GTM which is the essence of the *optimistic commit protocol* [19]. Compensating transactions have been initially introduced by Gray [15] and used as the basis for the Sagas [10] while Korth *et al* [17] studied the formal aspects of compensating transactions.

The second way of semantics compensation is through reservations. Reservations permits more interleaving between local and global transactions and at the same time work in situations where the classical compensation fails [23, 24].

4.2.3 Retry Approach

The retry approach is based on the assumption that a transaction will never get involved in a consistency violation when re-executed after a failure [16]. For example, in a banking system, a credit transaction will never violates consistency if there is no limits on the amounts of credits. Thus, such transactions will eventually commit if re-executed repeatedly. However, this approach has its limited applicability because transactions, in general, are not retriable. Also, notice that this approach differs from the redo approach since the whole transaction is executed again (including its read operations) but not only its write operations (which is the case in the redo approach). Thus, in the redo approach a transaction will move the database to a state that is equivalent to the sate had the transaction committed in its first execution whereas in the retry approach, this is not necessarily the case.

4.2.4 Hybrid Approach

The hybrid approach combines the three methods discussed above. For example, in the *pivot* method [21], a global transaction consists of a *pivot* subtransaction and a set of subtransactions where each of which is either compensateable of retriable. The pivot subtransaction is neither compensateable nor retriable. Furthermore, it is assumed that there is no value dependencies among the subtransactions (i.e., a write operation pertaining to a subtransaction is not a function of a read operation pertaining to another subtransaction). To ensure the atomicity of such a transaction, the compensateable subtransactions are executed first and committed. Then, the pivot is executed and committed. Finally, the retriable subtransactions are executed and committed. When the pivot is committed, the whole transaction has to commit. Otherwise, the whole transaction has to abort. Since each subtransaction before the pivot is compensateable, a compensating transaction for each of these transactions is launched if the pivot aborts. On the other hand, if the pivot commits, the retriable subtransactions are repeatedly executed until they all commit. A more general hybrid method that combines the undo, redo and retry approaches has been also proposed in the literature [5].

5 Unified ACPs

The methods that we discussed thus far complements each other. That is, a method that can be applied to one LDBMS might not be applicable to another in an MDBS. With minor differences with respect to their

classifications and terminologies, both Nodine [26] and Mullen [23] proposed a unifying approach in which they integrate the different methods that ensure the commitment of global transactions. Both researchers attempt to provide the most general and flexible framework that ensures the atomicity of transactions despite the diversity of the semantics of transactions and data.

6 Conclusion

In this paper, we surveyed and classified the different approaches that ensure the atomicity of global transactions in multidatabase systems. Our classification is based on the assumptions made by these approaches about the use of ACPs by the constituent database sites. We also discussed how the different approaches ensure the atomicity of global transactions. By relating the different methods to each other and highlighting the differences among them, our work should help in devising new and more flexible methods in this area of research.

References

[1] Al-Houmaily, Y. J. Commit Processing in Distributed Database Systems and in Heterogeneous Multidatabase Systems. Ph.D. Thesis, Department of Electrical Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania, April 1997.

[2] Al-Houmaily, Y. J. and P. K. Chrysanthis. Dealing with Incompatible Presumptions of Commit Protocols in Multidatabase Systems. *Proc. of the 11th ACM Annual Symposium on Applied Computing, Special Track on Database Technology*, pp. 186–195, February 1996.

[3] Barker, K. and M. T. Ozsu. Reliable Transaction Execution in Multidatabase Systems. *Proc. of the 1st Int'l Workshop on Interoperability in Multidatabase Systems*, pp. 344–347, April 1991.

[4] Braginski, E. The X/Open DTP Effort. *Proc. of the 4th Int'l Workshop on High Performance Transaction Systems*, Asilomar, California, September 1991.

[5] Breitbart, Y., H. Garcia-Molina and A. Silberschatz. Overview of Multidatabase Transaction Management. *VLDB Journal*, 1(2):181–239, October 1992.

[6] Breitbart, Y. and A. Silberschatz. Strong Recoverability in Multidatabase Systems. *Proc. of the 2nd Int'l Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, pp. 170–175, February 1992.

[7] Breitbart, Y., A. Silberschatz and G. Thompson. Reliable Transaction Management in a Multidatabase System. *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, pp. 215–224, May 1990.

[8] Breitbart, Y., A. Silberschatz and G. Thompson. Transaction Management Issues in a Failure-Prone Multidatabase System Environment. *VLDB Journal*, 1(1):1–39, July 1992.

[9] Chrysanthis, P. and K. Ramamritham. Autonomy Requirements in Heterogeneous Distributed Database Systems. *Proc. of the Conference on the Advances on Data Management*, pp. 283–302, December, 1994.

[10] Garcia-Molina, H. and K. Salem. Sagas. *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, pp. 249–259, 1987.

[11] Georgakopoulos, D. Transaction Management in Multidatabase Systems. Ph.D. Thesis, Department of Computer Science, University of Houston, December 1991.

[12] Georgakopoulos, D. Multidatabase Recoverability and Recovery. *Proc. of*

the 1st Int'l Workshop on Interoperability in Multidatabase Systems, pp. 348–355, April 1991.

[13] Georgakopoulos, D., M. Rusinkiewics and A. Sheth. On Serializability of Multidatabase Transactions Through Forced Local Conflicts. *Proc. of the 7th Int'l Conference on Data Engineering*, pp. 348–355, 1991.

[14] Gligor, V. and G. Lunckenaugh. Interconnecting Heterogeneous Database Management Systems. *IEEE Computer*, 17(1):33–43, January 1984.

[15] Gray, J. Notes on Data Base Operating Systems. In Bayer R., R.M. Graham, and G. Seegmuller (Eds), *Operating Systems: An Advanced Course*, *Lecture Notes in Computer Science*, Volume 60, pp. 393–481, Springer-Verlag, 1978.

[16] Hsu, M. and A. Silberschatz. Unilateral Commit: A New Paradigm for Reliable Distributed Transaction Processing. *Proc. of the 7th Int'l Conference on Data Engineering*, pp. 286–293, 1991.

[17] Korth, F., E. Levy and A. Silberschatz. A Formal Approach to Recovery by Compensating Transactions. *Proc. of the 16th Int'l Conference on Very Large Databases*, pp. 95–106, 1990.

[18] Lampson, B. Atomic Transactions. In *Distributed Systems: Architecture and Implementation - An Advanced Course*, B. Lampson (Ed.), *Lecture Notes in Computer Science*, Volume 105, pp. 246-265, Springer-Verlag, 1981.

[19] Levy, E., H. Korth and A. Silberschatz. An Optimistic Commit Protocol for Distributed Transaction Management. *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, pp. 88–97, 1991.

[20] Mehrotra S., R. Rastogi, Y. Breitbart, H. Korth and A. Silberschatz. Ensuring Transaction Atomicity in Multidatabase Systems. *Proc. of the ACM Symposium on Principles of Database Systems*, pp. 164–175, June 1992.

[21] Mehrotra, A., R. Rastogi, H. Korth and A. Silberschatz. A Transaction Model for Multidatabase System. *Proc. of the Int'l Conference on Distributed Computing Systems*, pp. 56–63, June 1992.

[22] Mohan, C., K. Britton, A. Citron and G. Samaras. Generalized Presumed Abort: Marrying Presumed Abort and SNA's LU 6.2 Commit Protocols. *Proc. of the 5th Int'l Workshop on High Performance Transaction Systems*, September 1993.

[23] Mullen, J. Atomic Commitment in Multidatabase Systems. Ph.D. Thesis, Department of Computer Science, Purdue University, December 1993.

[24] Mullen, J., J. Jing and J. Sharif-Askary. Reservation Commitment and its use in Multidatabase Systems. *Proc. of the 4th IEEE Int'l Conference on Database and Expert Systems Applications (DEXA)*, pp. 116-121, September 1993.

[25] Muth, P. and T. Rakow. Atomic Commitment for Integrated Database Systems. *Proc. of the 7th Int'l Conference on Data Engineering*, pp. 296–304, April 1991.

[26] Nodine, M. Interactions: Multidatabase Support for Planning Applications. Ph.D. Thesis, Department of Computer Science, Brown University, May 1993.

[27] O'Neil, P. The Escrow Transactional Method. *ACM Transactions on Database Systems*, 11(4), December 1986.

[28] Perrizo, W., J. Rajkumar and P. Ram. HYDRO: A Heterogeneous Distributed Database System. *Proc. of ACM SIGMOD Int'l Conference on Management of Data* pp. 32–39, May 1991.

[29] Pu, C., A. Leff and S. Chen. Heterogeneous and Autonomous Transaction Processing. *IEEE Computer*, 24(12):64–72,, December 1991.

[30] Skeen D. Non-blocking Commit Protocols. *Proc. of the ACM SIGMOD Int'l Conference on the Management of Data*. pp. 133–142, May 1981.

[31] Soparkar, N. and H. Korth and A. Silberschatz. Failure–Resilient Transaction Management in Multidatabases *IEEE Computer*, 24(12):28–36, December 1991.

[32] Tal, A. and R. Alonso. Integration of Commit Protocols in Heterogeneous Databases. *Proc. of the Int'l Conference on Information and Knowledge Management*, pp. 27–34, November 1992.

[33] Tal, A. and R. Alonso. Commit Protocols for Externalized–Commit Heterogeneous Databases. *Distributed and Parallel Databases*, 2(2):209–234, April 1994.

[34] Upton IV, F. OSI Distributed Transaction Processing, An Overview. *Proc. of the 4th Int'l Workshop on High Performance Transaction Systems*, Asilomar, California, September 1991.

[35] Veijalainen, J. and A. Wolski. Prepare and Commit Certification for Decentralized Transaction Management in Rigorous Heterogeneous Multidatabases. *Proc. of the 8th Int'l Conference on Data Engineering*, pp. 470–479, February 1992.

[36] Wolski, A. and J. Veijalainen. 2PC Agent Method: Achieving Serializability in Presence of Failures in a Heterogeneous Multidatabase. *Databases: Theory, Design, and Applications*, (Eds.) Rishe, N., S. Navathe and D. Tal, pp. 268–287, 1991.