1993

Extending Multidatabase Transaction Management Techniques to Software Development Environments

Aidong Zhang

Omran Bukhres

Report Number:
93-081

EXTENDIDNG MULTIDATABASE TRANSACTION MANAGEMENT TECHNIQUES TO SOFTWARE DEVELOPMENT ENVIRONMENTS

Aidong Zhang
Omran Bukhres

CSD-TR-93-081
December 1993

Extending Multidatabase Transaction Management Techniques to Software Development Environments

Aidong Zhang and Omran Bukhres
Department of Computer Science
Purdue University
West Lafayette, IN 47907 USA

Abstract

A multidatabase system integrates a set of autonomous local databases that can be accessed as a single unit. Such a multidatabase environment is actually a special case of a more general software development environment in which local components may be either database systems or file systems. This paper discusses the new issues that arise in such software development environments and presents some initial solutions for these issues by extending multidatabase transaction management techniques.

1 Introduction

A software development environment (SDE) is a distributed heterogeneous software system in which local components can be either database systems or file systems. These local systems originally ran in isolation to support their individual applications. It then became evident that more complex applications involving multiple systems could be supported through intersystem cooperation. Consider an example at BNR [4]. A group of engineers at corporate headquarters is responsible for maintaining switching-system quality. An application software package, the Statistical Analysis System (SAS), is used to analyze available data. From the SAS output, they develop performance and reliability graphs, which are stored in a DB2 database on an IBM mainframe. This information is then be used by design engineers to improve the switching-system design. The integration of these systems must, however, be accomplished without the disruption of local system autonomy.

Such an autonomy feature has been recognized and studied in multidatabase systems. A multidatabase system (MDBS) serves to integrate a set of local database systems at various locations (sites). The central concern of such an integration is the preservation of the local autonomy of the component database systems. Aspects of autonomy such as design, execution, and control have been studied in [7, 2, 5, 12]. MDBSs process two varieties of transactions. Each local transaction accesses a local database only and is submitted directly to a local database system. Global transactions, in contrast, may simultaneously access several local databases and are submitted to an integration phase, where they are parsed into a set of global subtransactions to be submitted to local database systems.

Thus, a SDE is a generalized case of an MDBS. The techniques developed in MDBSs may be extended to SDEs. The aspects of the integration on system and language designs have been studied in [11, 4]. In this paper, we investigate the application of multidatabase transaction management techniques to the decentralized software development environment. In such an environment, the conditions that can be placed on local sites must be more relaxed than those which may be in effect in a multidatabase environment. For example, there may be no concurrency control mechanisms in place at local sites, and prepare-to-commit states may also not be supported at local sites. We will discuss approaches that can be developed to ensure the correct execution of global and local applications in this less restrictive environment.

2 Decentralized Model

As software development environments usually deal with a large number of local components, a decentralized approach to integration is essential for SDEs. Such a decentralized approach provides a high degree of fault-tolerance, and the system can be easily extended to accommodate new local sites. We therefore anticipate that the decentralized design of global application management will become an important feature of SDEs, particularly of those systems integrating a large number of participating local software systems.

A SDE consists of a set of {local software systems, denoted LSS_i, for $1 \leq i \leq m$}, where each LSS_i comprises an autonomous software system on a set D_i of data at the local site LS_i and a global application manager (GAM). The GAM is decentralized on all machines participating in the SDE. Each LSS is associated with a GAM server (GS), and all machines participating in the SDE can run the GAM interpreter (GI). A global application is submitted by invoking a process of the GAM interpreter at its machine while a local application is submitted directly to a LSS. All machines are connected by a computer network. Figure 1 illustrates this architecture.

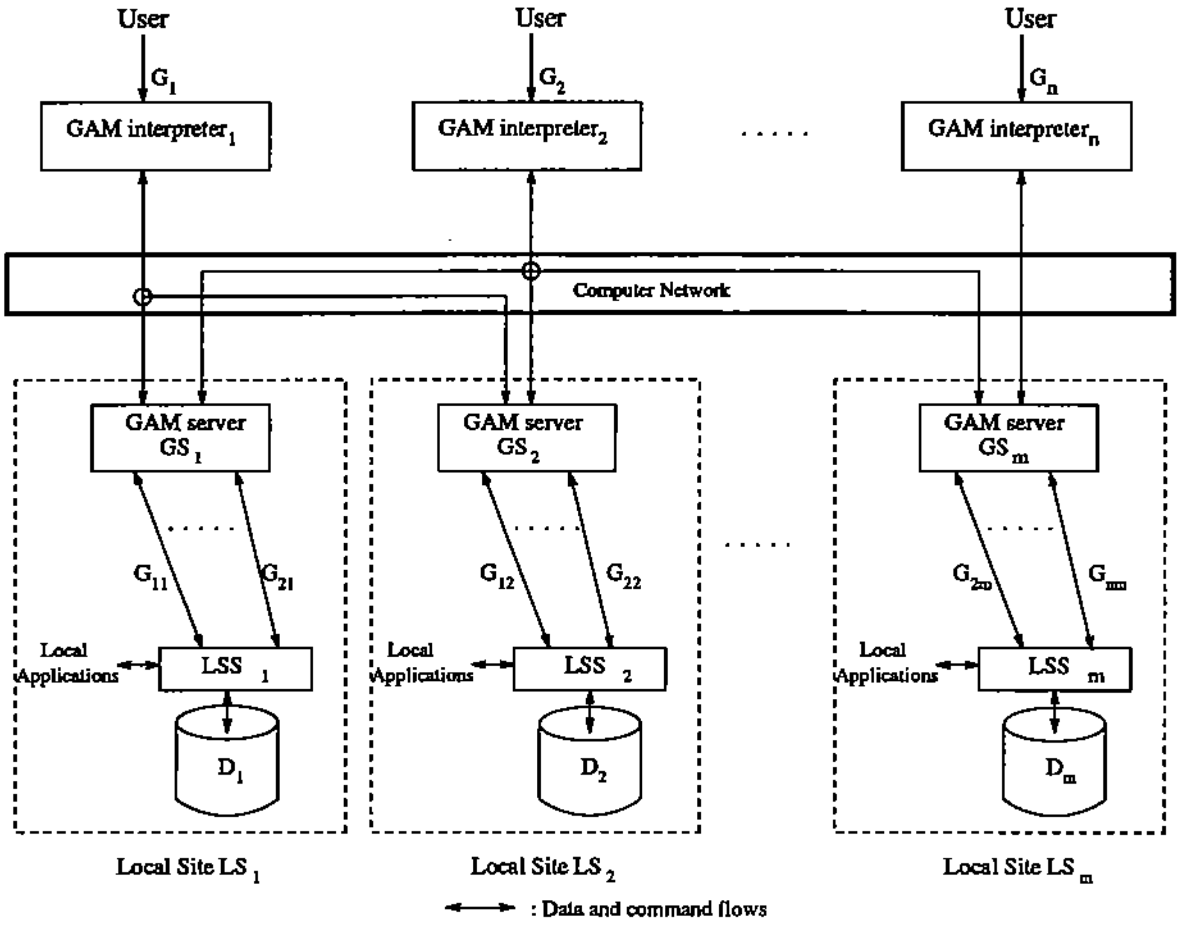

Figure 1: Decentralized SDE architecture

The GAM interpreter manages the decomposition and execution of global applications. In particular, the execution of a global application G_i is controlled by the GAM interpreter process GI_i, which submits the subapplications of G_i to the relevant GAM servers for execution. A GAM interpreter process can independently manage the execution of a global application without requiring any knowledge of the others' existence.

A GAM server is responsible for the execution of global subapplications received from the GAM interpreter processes. It then submits for execution each global subapplication to the LSS at its associated site. The completion of each subapplication is acknowledged by the LSS to the GAM server, which, if necessary, returns these results to the GAM interpreter processes. Each GAM server runs independently from other GAM servers and coordinates only with the GAM interpreter processes from which it receives global subapplications.

3 Global Concurrency Control Issues

Since some local software systems may not be database systems, they may not provide any concurrency control mechanisms. Thus, the GAM must be equipped to handle the mixed situation in which some local sites provide concurrency control mechanisms, while others do not. Without loss of generality, we assume that serializability[1] is ensured in those local sites which have concurrency control mechanisms.

A straightforward approach to controlling the execution of global applications is to ensure that all global applications run serially at the global level. In order to ensure the serial execution order of global applications at the global level, we must ensure that the execution orders of global subapplications at all local sites are *relatively synchronized*. That is, there exists a total order O on all global applications such that, at each local site, the execution order of its global subapplications is consistent with O. For those local sites which have no concurrency control mechanisms, both local applications and global subapplications must be executed serially. The execution order of global subapplications at the global level is obviously inherited at such local sites, and the effect of the execution of global subapplications at the global level is equivalent to the effect of their execution at these local sites.

It has been pointed out in [2, 5, 8] that the execution order of global subapplications at the global level may not be the same as their serialization order at a local site. Thus, even if the execution orders of global subapplications at all local sites are relatively synchronized, their execution orders at those local sites which have concurrency control mechanisms may not be identical to their serialization order. Consequently, the effect of the execution of global applications may not be serial. The enforcement of local conflicts [8, 13] has been proposed to force the execution order of global subapplications at the global level to conform to their serialization order at local sites.

The remaining issue in the implementation of an efficient decentralized scheme for the serial execution of global applications is the synchronization of the relative execution orders (REOs) of global subapplications at all local sites. Our method begins by numbering all GAM servers in an order O, with each GAM server maintaining a site-lock. Prior to executing global application G_i, GAM interpreter GI_i must first request all necessary site-locks from the relevant GAM servers in an order consistent with O. The REO of G_i is determined at all relevant sites only when GI_i has acquired the necessary site-locks. After the REO of G_i is determined, GI_i releases all held site-locks. During this process, if failures occur,

[1]In this paper, serializability refers to conflict serializability [1].

GI_i will request all relevant GAM servers to remove G_i from the pre-determined REOs and release all held site-locks. Because the site-locks are requested in an order consistent with O and the REO of G_i is determined only after GI_i holds all necessary site-locks, the correct synchronization of concurrent site-locks requests is ensured and correct REOs of global applications at all sites are thus guaranteed. After the REO of G_i is determined, GI_i sends subapplications of G_i to the relevant GAM servers. Using the ticket method [8], these GAM servers can enforce conflicts among all subapplications at each local site where a concurrency control mechanism is used and submit the subapplications for execution according to the pre-determined REO. This synchronization approach permits each global application to be submitted for execution without waiting for the completion of a previously submitted global application.

At this point, it is not clear to us whether more liberal approaches to controlling the execution of global applications at the global level will improve their execution performance. With the mixed situation in which some local sites provide concurrency control mechanisms, while others do not, those global subapplications which can be executed concurrently at some local sites may need to wait for the results of others which must be executed serially at other local sites. Depending on the load of data flow among global subapplications of each global application, those global correctness criteria, that permit global subapplications to run concurrently at the local sites which provide concurrency control mechanisms, may achieve better performance on the execution of global applications. Further investigation need be done to explore the effects of local concurrency control mechanisms on global concurrency control in SDEs.

4 Non-conventional Commit Protocols

The global applications in SDEs may be more flexible than global transactions in MDBSs. In other words, some actions in a global applications may not have to be executed or may be replaced by other actions if they fail. For example, in a debugging global application, the failure (existing bugs) of one source code module may be replaced by other version of the same module and also, partial execution of all source code modules may be acceptable. This flexibility allows a global application to adhere to a weaker form of atomicity, which we term *semi-atomicity*, while still maintaining its correct execution in the SDE environment. Semi-atomicity allows a global application to commit even if some subapplications fail, provided that either the failed subapplications are unnecessary to be executed or their alternative subapplications complete.

We are currently investigating non-conventional commit protocols for ensuring the semi-atomicity of global applications. To preserve the semi-atomicity of global applications, given the assumption that LSSs can recover from failures, the GAM must ensure either that all global subapplications of a global application that must complete commit or that none of the effects of each global subapplication remain permanent.

Preserving the atomicity [1] of global transactions in multidatabase systems has been recognized as an open and difficult issue [10]. Of particular concern is the fact that multidatabases cannot assume that the local databases support a visible prepare-to-commit state for those subtransactions in which a subtransaction has not yet been committed but is guaranteed the ability to commit. This scenario clearly remains problematic in the SDE environment. In such situations, a local software system that participates in a SDE environment may unilaterally fail a global subapplication without agreement from the global level (termed *a local unilateral fail*). As a result, it becomes difficult to ensure that a single complete logical action of the subapplications in a global application that must complete is consistently carried out at multiple local sites. The traditional two-phase commit (2PC) protocol developed in distributed database environments thus becomes inadequate to the preservation of the semi-atomicity of global applications in the SDE environment.

Both forward and backward recovery approaches which utilize the redo, retry, and compensation techniques have been proposed [3, 9] for the preservation of the semantic atomicity [6] of global transactions. These techniques allow each global subtransaction to commit unilaterally, requiring either the redo or retrial of aborted global subtransactions or the undoing of tentatively committed global subtransactions by corresponding compensating transactions. Note that at most one subtransaction of each global transaction can be pivot. The compensatable subtransactions must be completed before the completion of the pivot subtransaction, which in turn must complete before the completion of the retriable subtransactions. The global complete/fail decision is determined by the outcome of the completion of the pivot subtransaction. If it fails, all of the compensatable subtransactions are compensated for; otherwise the retriable subtransactions are attempted until they complete.

Semi-atomicity is also weaker than semantic atomicity. In order to utilize the above techniques in the SDE environment, we categorize each global subapplication as either *retriable*, *compensatable*, or *pivot*. We say that a subapplication is *retriable* if it is guaranteed to commit after a finite number of submissions when executed from any consistent database state. A subapplication is *compensatable* if the effects of its execution can be semantically undone after commitment by executing a compensating subapplication. A compensating subapplication ct_i for a subapplication t_i must be independent of the applications that execute between t_i and ct_i. This is because local database autonomy requires that arbitrary local applications be executable between the time t_i is committed and the time ct_i is executed, and these local applications can both see and overwrite the effects of t_i during that time. A subapplication is a *pivot* subapplication if it is neither retriable nor compensatable.

Our investigation starts with the construction of global applications whose semi-atomicity can be maintained in the SDE environment. In general, the semi-atomicity of a global application may not be ensured without using local prepare-to-commit states. For example, the possession of two or more pivot subapplications that must be executed in a global application may render it difficult to determine a commit order among them which ensures that the global application can move either forward to the commitment of its subapplications or backward to the removal of any partial effects of the committed subapplications. However, if one pivot subapplication can be replaced by some other retriable subapplication, then the global application may still proceed when the pivot subapplication fails. The establishment of the essential properties of global applications such that their semi-atomicity can be ensured provides a foundation for the design of an advanced global application language. We are also investigating the decentralized commit protocol for ensuring the semi-atomicity of such global applications.

5 Conclusions

In this paper, we have discussed those issues that arise when the techniques of multidatabase transaction management are applied to the software development environment. Our investigation represents a first step toward global application management in SDEs, one which may be amplified in the future by additional refinement. The synchronization approach described above has been implemented in the context of our InterBase project, the design of the commit protocol is currently being investigated as part of that project.

References

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

[2] Y. Breitbart and A. Silberschatz. Multidatabase Update Issues. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 135–142, June 1988.

[3] Y. Breitbart, A. Silberschatz, and G. Thompson. Reliable Transaction Management in a Multidatabase System. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 215–224, May 1990.

[4] O. A. Bukhres, J. Chen, W. Du, A. K. Elmagarmid, and R. Pezzoli. InterBase: An Execution Environment for Heterogeneous Software Systems. *IEEE Computer*, 26(8):57–69, August 1993.

[5] W. Du and A. Elmagarmid. Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase. In *Proceedings of the 15th International Conference on Very Large Data Bases*, pages 347–355, Amsterdam, The Netherlands, Aug. 1989.

[6] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Trans. Database Syst.*, 8(2):186–213, June 1983.

[7] H. Garcia-Molina and B. Kogan. Node Autonomy in Distributed Systems. In *Proceedings of the First International Symposium on Databases for Parallel and Distributed Systems*, pages 158–166, Austin, Texas, USA, Dec. 1988.

[8] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. On Serializability of Multidatabase Transactions Through Forced Local Conflicts. In *Proceedings of the 7th Intl. Conf. on Data Engineering*, pages 314–323, Kobe, Japan, Apr. 1991.

[9] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. A transaction model for multidatabase systems. In *Proceedings of International Conference on Distributed Computing Systems*, June 1992.

[10] N. Soparkar, H. F. Korth, and A. Siberschatz. Failure-resilient transaction management in multidatabases. *IEEE Computer*, 24(12):28–36, December 1991.

[11] P. Tarr and S. Sutton. Programming heterogeneous transactions for software development environments. In *Proceedings of Fifteenth International conference on Software Engineering*, pages 358–369, 1993.

[12] J. Veijalainen. *Transaction Concepts in Autonomous Database Environments.* R. Oldenbourg Verlag, Germany, 1990.

[13] A. Zhang and A. K. Elmagarmid. A theory of global concurrency control in multidatabase systems. *The VLDB Journal*, 2(3):331–359, July 1993.