Management of Interdependent Data: Specifying Dependency and Consistency Requirements

Amit Sheth
Marek Rusinkiewicz

Bellcore
444 Hoes Lane, Piscataway, NJ 08854

1 Introduction

Like many large companies, Bellcore uses multiple databases that serve the needs of various application systems. One of the significant problems in managing these databases is to maintain the related data items consistent to the required degree. This problem is frequently referred to as "redundant data management". Since the term "redundancy" tends to imply that the data is unwanted or superfluous, we will use here a more general term "management of interdependent data".

In this paper, we discuss some preliminary ideas in this area[1] First, we characterize the problem of managing interdependent data and identify some important types of interdatabase dependency. Then we define the types of consistency requirements which can exist among the interdependent data. Finally, we comment on the previous work on transaction management reported in the literature and its relevance to our problems.

2 Characterizing the Problem of Interdependent Data Management

To analyze the problem we first formulate a set of basic criteria that can be used to characterize the relationships between the data stored in various data repositories. These criteria create a taxonomy which will allow us to define the problem more precisely and find solutions suitable for a particular subclass of problems.

(a) **Type of Interdatabase Dependency**
What kinds of dependencies exist among the data stored in multiple systems? For example, the data may be fully or partially replicated, the informational content of the data may overlap, a database may store information derived from one or more other databases, etc. Interdatabase dependency may also exist because of the constraints specified by data definition (e.g., fragmentation, interdatabase constraints) or through application programs.

(b) **System Heterogeneity**
What is the degree of system heterogeneity among various repositories of data?[2] What transaction support and recovery primitives are provided by a particular local system? (Explicit locking? Visible Prepared-to-commit state? Rollback? Checkpointing?)

(c) **Degree of Local Autonomy**
What is the degree of coupling between the data repositories involved? This can range from a tight functional coupling, through various level of federation, to almost complete design and execution autonomy.

(d) **Data Consistency Criteria**
What is the desired degree of consistency among data? Should the data in multiple repositories be mutually consistent at every point of time? Can this requirement be relaxed so that the consistency of data is restored with some delay? Does every transaction need to see an up-to-date view of the data or maybe a consistent snapshot with some guaranteed delay is sufficient?

Various areas in the space defined by the above dimensions define various subclasses of redundant data

[1] The views expressed in this paper are those of the authors and are not necessarily the views of Bellcore.

[2] The system heterogeneity is different from *semantic heterogeneity* that results from the fact that the data stored in various repositories may not directly correspond to each other because of the differences in underlying data definitions.

TH0329-3/90/0000/0133$01.00 © IEEE

management problem. Depending on this, various solutions in the area of transaction management (including commitment, concurrency control and recovery) and various correctness criteria may become applicable.

3 Types of Interdatabase Dependency

Two aspects can be used to characterize relationships among various databases or copies of data: structure of dependency and control (or ownership).
The **structural dependencies** among the related data include the following relationships:

(a) **Replicated Data:** Identical copies of data items are stored in two or more databases.

(b) **Vertical and Horizontal Partitions:** Data are logically related but physically fragmented and stored in different databases. This can be seen as a frequently occurring special case of a constraint dependency, defined below.

(c) **Constraint Dependency:** Data in one database may be related to the data in another database by various types of value dependency or constraints (e.g., referential integrity or numerical consistency).

The **control dependencies** among related data include the following relationships:

(a) **Derived Data:** Here, there exists a *source database* from which data is derived and stored in other databases. This derivation may be an extraction of selected data or an aggregation of the data from the source database. No direct updates are performed on the derived data and the updates to the source database do not need to be propagated immediately to the derived data.

(b) **Primary-Secondary Copies:** Here the updates on the primary database have to be applied to the secondary databases, so that the secondary databases achieve consistency with the primary database when these updates are performed. A coordinator-subordinate relationship exists between the primary and secondary databases. It is understood that only the primary database contains correct (current) data.

(c) **Interdependent Databases:** In this case all the databases are peers that store interrelated data. Thus, updates to one database often implies that related updates have to be applied to the others, so that the interdatabase integrity is preserved.

Data Consistency Criteria

The consistency among interrelated data is called mutual consistency. Most of the work done on management of replicated data used one copy serializability to assure full mutual consistency at all times. Lately, researchers and developers have been looking at the weaker criteria for mutual consistency. We propose that the mutual consistency can be specified along two dimensions: time and space.
Temporal Component of Mutual Data Consistency

One copy serializability and other methods using multi-site transactions which commit updates at multiple sites provide *immediate consistency* of the related data. The idea of specifying time when the consistency of related data must be restored was introduced in [WQ87] where the concept of *eventual match* is discussed. The following kinds of consistency along the time dimension can be identified [SK89]:

- **Eventual Consistency** requires that redundant copies are consistent at certain times, although they may be not consistent in the interim intervals. The times when the redundant copies are consistent may be specified in several ways: (a) at specified time intervals, e.g., the redundant copies will be consistent every one hour, (b) at specified time point, e.g., the redundant copies will be made consistent at 5 PM every Friday, and (c) at a specified event, e.g., the redundant copies will be made consistent before running a specified transaction, or when a particular command will be given to the system. This criterion allows the redundant data to diverge during some period, as long as the copies are made consistent periodically.

- **Lagging Consistency** assumes that the data in one copy may be most current while in other copies, the data may not be up-to-date. Updates applied to one copy are always propagated to the redundant copies. Hence, if all external updates are stopped, the redundant copies will become consistent. This can be seen as a degenerate case of eventual consistency. Eventual consistency does imply that at some point in time, all redundant data will be consistent, while lagging consistency does not imply this, because some copies of data may always lag behind more current copies.

Both these criteria allow redundant copies to be temporarily inconsistent and hence allow delayed updating of one or a few of the redundant copies. Of course, each copy is individually consistent. Some of the situations where such criteria may be acceptable include: (a) an application looks at only one of the copies at a time, or (b) the database management software always directs the application query to the most consistent copy, or (c) the application does not require the most current and consistent data.

Spatial Component of Mutual Data Consistency

The idea of specifying "how far" the related data may be allowed to diverge before the mutual consistency must be restored was discussed in [ABG88] where the concept of *quasi copy* is introduced. The spatial component of the mutual data consistency can be specified in one of the following ways:

(a) By specifying (or limiting) the **number of data items** that can be changed before the consistency must be restored. As an example, let us consider a database providing statistical information that is derived by aggregation of individual data records. We may specify that the statistical data must be recalculated whenever a given percentage of the individual data records is updated.

(b) By specifying (or limiting) the maximum **change in the value of data** before the consistency must be restored. As an example, let us consider an inventory balance which is stored in one database while the sales transaction records are recorded in another database. We may specify that the inventory (which is used, for example, to re-order new shipments) does not need to be updated if the individual sale does not exceed 100 items.

(c) By specifying (or limiting) the **number of operations** allowed on any interrelated data during the period the related data are inconsistent. In the previous example, we may allow no more than 10 sales transactions before the inventory in another database is updated.

4 Relation to Previous Work

Several solutions have been proposed to manage interdependency (redundancy) in distributed databases in literature. Most of the proposed solutions assume the following regarding the system environment and consistency requirements:

1. The interdependent data appear in exactly the same form, i.e., as replicas in multiple databases.
2. The databases are well integrated, i.e., the model of the system is either that of a distributed DBMS or of a tightly coupled system.
3. The consistency criterion is providing immediate consistency of replicas by using *one copy serializability* of update transactions.
4. All systems participating in the distributed DBMS are identical and equally reliable (also called the assumption of interchangeable systems).

Many solutions in the area of database transaction management have been proposed assuming the environments such as the one defined above, which limits their applicability. Many systems used in Bellcore do not satisfy one or more of the above requirements. Similarly, some of the transaction management solutions proposed for multidatabase systems (e.g. quasi-serializability [DE89]), may not be directly applicable for managing mutual consistency among interdependent data, because of the required assumptions. Hence, new solutions suitable for various classes of systems that can be defined using the taxonomy introduced earlier, must be sought.

The main contributions of this work are: (a) definition of a taxonomy allowing to define various subclasses of the problems of interdependent data management, (b) preliminary classification of the types of interdatabase dependencies, and (c) definition of various data consistency criteria with temporal and spatial dimensions. We are currently working on identifying and developing solutions that are suitable for various subclasses of the problems identified above.

Acknowledgements

Prabhakar Krishnamurthy and Arbee Chen participated in the work on redundant data management at Bellcore.

References

[*ABG88*] Rafael Alonso, Daniel Barbara, and Hector Garcia-Molina,"Quasi-Copies: Efficient Data Sharing for Information Retrieval Systems," In *Proceedings of the Intl. Conf. on Extending Data Base Technology*, (Venice, Italy, Mar.), (Published as Lecture Notes in Computer Science, Vol 303, Springer-Verlag).

[*DE*89] W. Du and A. Elmagarmid, "Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase," *Proc. of the Intl. Conf. on Very Large Databases,*" Amsterdam, August 1989.

[*SK*89] A. Sheth and P. Krishnamurthy, "Redundant Data Management in Bellcore and BBC Databases," *Technical Report TM-STS-015011/1*, Bellcore, December 1989.

[*WQ*87] G. Wiederhold and X. Qian, "Modeling Asynchrony in Distributed Databases," *Proc. of the Intl. Conf. on Data Engineering*, February 1987.