Transactions across Heterogeneous Databases: the Superdatabase Architecture

Calton Pu

Department of Computer Science
Columbia University
New York, NY 10027

Technical Report No. CUCS-243-86
Revised - June 1988

Abstract

We propose the superdatabase architecture to support atomic transactions across heterogeneous databases. We start with hierarhical composition of element databases, followed by optimization and distribution. Heterogeneous crash recovery translates different commit agreement protocols. Heterogeneous concurrency control groups different kinds of algorithms. such as two-phase locking, timestamps, and optimistic methods to preserve transaction concurrency, and certifies the subtransactions from participating groups at transation commit. The superdatabase consumes very little run-time overhead and few messages. A prototype element database, called Nova, and a prototype superdatabase, called Supernova, are currently under construction at Columbia University.

[0]This work has been partially supported by New York State Center for Technology in Computer and Information Systems under grant Number NYSSTF-87(5) and a Digital Equipment Corporation equipment grant.

Contents

1 Introduction

For both efficiency and extensibility, integrated and consistent access to a set of heterogeneous databases is desirable. However, current commercial databases running on mainframes are, by and large, centralized systems. R* [16] and INGRESS/STAR [19] have demonstrated physical distribution of homogeneous databases. Still, research on integrated heterogeneous databases has been largely limited to query-only systems such as MULTIBASE and MERMAID.

MULTIBASE [14] is a retrieve-only system, developed at Computer Corporation of America. Through the DAPLEX functional language, MULTIBASE provides uniform access to a CODASYL database and a hierarchical database. The focus of MULTIBASE is on query optimization and reconciliation of data, and consistent access across databases were not part of their goals. MERMAID [28] has been developed at System Development Corporation. Unlike MULTIBASE, MERMAID supports the relational view of data directly, through the ARIEL query language, a superset of SQL and QUEL. Another project providing a common query language to access databases using different data models is SIRIUS-DELTA [9].

Complementing earlier works on uniform query access, our research concentrates on consistent update across heterogeneous databases. A *superdatabase* is the glue that supports atomic transactions across heterogeneous element databases, which may be centralized, distributed, or another superdatabase.

Update support in homogeneous databases relies on two sets of fundamental techniques: concurrency control and crash recovery. We are building the superdatabase through hierarchical composition of concurrency control and crash recovery, followed by optimziation and distribution. Many years of research on nested transactions [17,20,25] have produced specific algorithms to implement nested transactions organized into a hierarchy. In the Eden system [23], we have applied systematic hierarchical composition to derive the design and implementation of a nested transaction mechanism. In the preliminary version of this paper [24], we concentrated on hierarchical algorithms. Here, we optimize transaction concurrency and execution overhead through grouping of concurrency control methods and flattening of hierarchical recovery algorithms.

In Section 2 we describe the algorithms of hierarchical composition. Section 3 summarizes the optimization techniques to improve superdatabase performance and transaction concurrency. Section 4 outlines current implementation status. In Section 5 we summarize related work on many different aspects of heterogeneous databases. Finally, Section 6 concludes the paper.

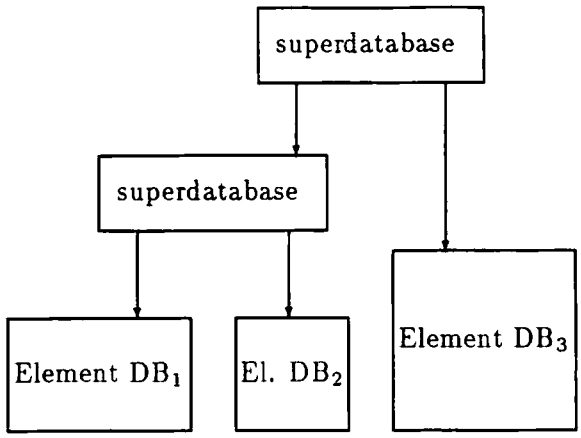

Figure 1: Conceptual Structure of Superdatabases

2 Hierarchical Composition

2.1 Tree-Structured Superdatabase

We start by establishing terminology. An *element database* is a database manager with its own concurrency control and crash recovery to support atomic transactions. A superdatabase is the glue that implements atomic transactions across different element databases. In figure 1, DB_j (the leaves) represent different element databases glued together by superdatabases (the internal nodes). A transaction spanning several element databases is called a *supertransaction* [21]. When participating in a supertransaction, the local transaction on each element database is called a subtransaction. We assume a supertransaction is translated into no more than one subtransaction for each element database. This is a standard assumption [11].

Two reasons motivated this initial tree-structured organization. First, hierarchical organization minimizes the amount of data transfer in both size and number of messages. For instance, in section 2.4 we show that we only need to piggyback a small amount of information on messages already required for distributed commit protocols. Second, hierarchical algorithms are easy to explain and understand. In section 3 we will introduce the optimizations to improve performance and concurrency, in addition to distribution for availability.

For hierarchical composition, An element database is *composable* if it satisfies two requirements. The first is on crash recovery: the element database must understand some kind of agreement protocol, e.g. two-phase commit. As we shall see in section 2.2.1, this requirement

is a necessary consequence of distributed control, not heterogeneity. The second requirement is on concurrency control: the element database should present an explicit serial ordering of its local transactions. For concurrency control methods with explicit ordering (e.g. basic timestamps) this requirement is trivial. In general, the proof of serializability for concurrency control algorithms usually provides us with a way to explicitly capture the serial order they impose on the transactions.

For consistent updates, these two are the only requirements we make on the element databases.[1] An element database may be centralized, distributed, or another superdatabase. Since centralized databases do not need agreement protocols nor do they supply the transaction serial order, superdatabase cannot glue existing databases "as is" together. Nevertheless, we believe that these requirements, mild for distributed databases, can be feasibly incorporated into current and future database systems. The pay-off is significant: extensibility and accommodation of heterogeneity.

We have three design goals for the superdatabase that glues the composable element databases together to support consistent update across heterogeneous element databases:

1. Composition of element databases with different crash recovery methods.

2. Composition of element databases with different concurrency control techniques.

3. Recursive composibility; i.e. the superdatabase should be composable.

The realization that we need only an agreement protocol for crash recovery made the first goal easy. The simple idea that achieved the second goal is to use the explicit serial ordering of transactions, the common denominator of best known concurrency control methods. The third goal was accomplished through careful design of the agreement protocol and explicit passing of the serial order.

2.2 Hierarchical Recovery

2.2.1 Heterogeneous Hierarchical Commit

The usual model of a distributed transaction contains a coordinator and a set of subtransactions. Each subtransaction maintains its local undo/redo information. At transaction commit time, the coordinator organizes some kind of agreement with subtransactions to reach a uniform decision. Without agreement protocols, one subtransaction may commit while another aborts. Therefore, the need for agreement on the transaction outcome is due

[1] We should note that the commit protocol adds some complication to the recovery algorithms in centralized databases.

to distribution, not heterogeneity. Two-phase commit is the most commonly used protocol for its low message overhead.

The distributed database system R* [16] provides a tree-structured model of computation that refines the above flat coordinator/subtransactions model. Subtransactions in R* are organized in a hierarchy, and the two-phase commit protocol is extended to the tree structure. At each level, the parent transaction serves as the coordinator. During phase one, the root sends the message "prepare to commit" to its children. The message is propagated down the tree, until a leaf subtransaction is reached, when it responds with its vote. At each level, the parent collects the votes; if all its own children vote "yes", then it sends "yes" to the grand-parent. If every subtransaction voted "yes", the root decides to commit and sends the "committed" message, again propagated down the tree. Between the sending of its vote and the decision by the root, each child subtransaction remains in the *prepared* state, ready to both undo the transaction if aborted, and to redo the transaction if the child crashed and the root decided to commit.

Since heterogeneous databases are distributed by nature, each element database maintains the undo/redo information locally. The superdatabase stores only global transaction management information and relies on element databases for local recovery. In addition, each element database must understand some kind of agreement protocol, such as the two-phase commit outlined above, three-phase commit, or the various flavors of Byzantine agreements.[2]

Given that some form of agreement is necessary due to distribution, the question is whether it is sufficient for hierarchical commit. R* implements hierarchical two-phase commit. Three-phase commit and Byzantine agreements also have natural extensions to tree-structured computations. The important fact is that for each element database, the superdatabase must understand and use the appropriate protocol. If all element databases use the same protocol, the superdatabase has the obvious role in the hierarchical protocol. Interesting cases arise when element databases support different kinds of agreement protocols.

To simplify the discussion, we divide the distributed agreement protocols into two groups: symmetric and asymmetric. Symmetric protocols such as Byzantine agreements and decentralized two-phase commit give all participants equal role. In asymmetric protocols, a distinguished coordinator decides the outcome based on information supplied by other participants. For example, in the centralized and linear two-phase commit, as well as the three-phase commit, a coordinator initiates the protocol and decides whether the transaction commits or aborts.

[2] In the discussion below, references on the Byzantine agreements can be found in several PODC Proceedings; the other protocols are described in the recent book by Bernstein et al. [3].

If an element database supports an asymmetric agreement protocol, the superdatabase assumes the role of coordinator with respect to that element database. Notice that the superdatabase may have to act as the coordinator for different protocols. Since the decision is made by the coordinator in all asymmetric protocols, the superdabase collects information and decides.

If some element databases employ symmetric protocols, we have two choices for the superdatabase. First, the naive method simulates the symmetric protocol for all element databases by translating the information received from "asymmetric" element databases and passing it to the "symmetric" participants. For example, consider three element databases, DB_1 and DB_2 with two phase commit and DB_3 using symmetric two-phase commit. In this naive method, the superdatabase passes the votes from DB_1 and DB_2 to DB_3 explicitly, even though the knowledge of the existence of DB_1 and DB_2 does not increase system resiliency to crashes, since DB_1 and DB_2 do not know symmetric two-phase commit to help DB_3 recover if the superdatabase crashes. This method makes it easy to prove the correctness of the combined algorithm, but sends unnecessary messages.

Second, an optimized superdatabase may eliminate the extra messages by serving as a representative of the "asymmetric" participants, sending the result of the asymmetric protocols in one round of messages. This second method decreases the number of messages by combining the extra messages into one. These two choices also exist for the communication between "symmetric" participants using different protocols. The message savings for m participants of one protocol and n of the other is $(m \times n) - (m + n)$.

Between a superdatabase and its parent, we can use any agreement protocol that both understand. In this paper and in our implementation, we adopt two-phase commit to minimize message overhead.

In summary, the superdatabase functions both as a coordinator for the asymmetric agreement protocols and as a translator for the symmetric protocols. It collects sufficient information for supertransaction commit, and provides enough information for participants using symmetric protocols to reach their own conclusion that matches the superdatabase's.

2.2.2 Superdatabase Recovery

Since the superdatabase is the coordinator for the element databases during commit protocols, it must record the transaction management information on stable storage. Otherwise, a crash during the window of vulnerability would hold resources in the element databases indefinitely.

Of the known recovery methods, logging is the best for superdatabase recovery. Since no

before-images or after-images need to be saved, versions are of little utility. Conceptually, the superdatabase log is separate from the element database logs, just as the superdatabase itself. In actual implementation, the superdatabase log may be physically interleaved with an element database log, as long as the recovery algorithms can separate them later.

For each transaction, the superdatabase saves the following information on the log:

- Participant subtransactions.
- Parent superdatabase, if any.
- Transaction id and state (active, prepared, committed, or aborted).

The superdatabase should remember the participant subtransactions because the supertransaction does not necessarily abort when the superdatabase crashes. Suppose that the superdatabase crashes, but is brought back online quickly, before its subtransactions have finished. Since the superdatabase performs no computation, the supertransaction may still commit. To carry out commit agreement after such glitches, the participant subtransactions should be remembered in the log, which is read at restart time to reconstruct the superdatabase state before the crash.

The transaction state is written to the log during the commit agreement. If a transaction was in the active state when the superdatabase crashed, the superdatabase simply waits for (re)transmission of two-phase commit from the parent. In case it is the root, it (re)starts the two-phase commit. If a transaction was in the prepared state when the superdatabase crashed, the superdatabase inquires the parent about the outcome of the transaction. If the transaction has been committed, the results are retransmitted to the subtransactions.

2.3 Hierarchical Concurrency Control

2.3.1 Heterogeneous Concurrency Control

To motivate the certification algorithm to serialize hetergeneous transactions, let us consider the following example. The supertransaction T_1 has subtransactions $T_{1.1}$ and $T_{1.2}$ running on element databases DB_1 and DB_2, respectively:

```
BeginTransaction(Top-level, tid: T1)
    cobegin
        DB1.BeginTransaction(parentid: T1, tid: T1.1)
            ... actions ... DB1.CommitTransaction(T1.1)
        DB2.BeginTransaction(parentid: T1, tid: T1.2)
```

... *actions* ... DB_2.CommitTransaction($T_{1.2}$)

coend

CommitTransaction

Suppose DB_1 and DB_2 use two-phase locking. If $T_{1.1}$ starts releasing locks while $T_{1.2}$ has not reached its lock point, the supertransaction T_1 may lose its two-phase property and become non-serializable. This scenario reveals the crucial problem in hierarchical composition of concurrency control mechanisms: local serialization does not always guarantee global serialization. Therefore, in the general case the superdatabase should certify that all local serial orders are compatible in a global serial order. One way to implement the superdatabase certification is to require that each element database provide the ordering of its local transactions to the superdatabase. This method is sufficient for composition of heterogeneous databases, but not necessary, since implicit serialization is possible under certain circumstances (section 3.2). The serial order of each local transaction is represented by an *order-element*, or O-element for short. Before we describe the composition of O-elements for certification in section 2.3.2, we first discuss how the concurrency control methods produce the O-elements.

First, we consider element databases with two-phase locking concurrency control. Transactions using two-phase locking acquire all locks in a growing phase, and then release them during a shrinking phase, in which no additional locks may be acquired. Eswaran et al. [8] showed that two-phase locking guarantees serializability of transactions because $SHRINK(T_i)$, the timestamp of transaction T_i's lock point, indicates T_i's place in the serialization. We designate $SHRINK(T_i)$ as the O-element for element databases with two-phase locking.

Second, timestamps used for serialization represent an explicit ordering, so they serve well as O-elements. Timestamp intervals [2] or multidimensional timestamps [15] can be passed as O-elements as well. The important thing is to capture the serialization order of committed local transactions.

Third, optimistic concurrency control methods also provide an explicit serialization order. Kung and Robinson [12] assign a serial transaction number after the write phase, which can be used directly as O-element. Ceri and Owicki [6] proposed a distributed algorithm in which a two-phase commit follows a successful validation. Taking a timestamp from a Lamport-style global clock [13] at that moment will capture the serial order of transactions. Since the write phase has yet to start, all following transactions will have a later timestamp.

There is no constraint on the format of the O-element. Each element database may have its own representation. We only require that two O-elements from the same element database be comparable, and that a comparison recover the serialization order guaranteed

by local concurrency control methods. More formally, let the serialization produced by the concurrency control method be represented by the binary relation *precede* (denoted by $\prec$), if O-element(T_1) $\prec$ O-element(T_2) then $T_1 \prec T_2$ in the local serialization.

2.3.2 Hierarchical Certification with O-vectors

The main problem that the superdatabase has to detect is when subtransactions from different element databases have been serialized in different ways. In our example, this happens when a second transaction T_2 with the same subtransactions produces the ordering:

O-element($T_{1.1}$) $\prec$ O-element($T_{2.1}$) and
O-element($T_{2.2}$) $\prec$ O-element($T_{1.2}$).

To detect this kind of disagreement, we define an order-vector (O-vector) as the concatenation of all O-elements of the supertransaction. In the example, O-vector(T_1) is (O-element($T_{1.1}$), O-element($T_{1.2}$)). The order induced on O-vectors by the O-elements is defined strictly: O-vector(T_1) $\prec$ O-vector(T_2) if and only if for all element database j, O-element($T_{1.j}$) $\prec$ O-element($T_{2.j}$). If a supertransaction is not running on all element databases, we use a wild-card O-element, denoted by * (star), to fill in for the missing element databases. Since its order does not matter, by definition, O-element(any) $\prec$ *, and, * $\prec$ O-element(any).

From this definition, if O-vector(T_1) $\prec$ O-vector(T_2) then all subtransactions are serialized in the same order, serializing the supertransactions. Therefore, we can serialize the supertransactions by checking the O-elements of a committing supertransaction against the history of all committed supertransactions. If the new O-vector can find a place in the total order, it may commit.

From the composition point of view, the key observation is that the certification based on O-vectors is independent of particular concurrency control methods used by the element databases. Therefore, a superdatabase can combine heterogeneous concurrency control methods. As long as we can make the serialization in element databases explicit, the superdatabase can certify the serializability of supertransactions.

Equally important, the certification gives the superdatabase itself an explicit serial order (the O-vector) allowing it to be recursively composed as an element database. Thus we have found a way to hierarchically compose database concurrency control, maintaining serializability at each level.

The certification method is optimistic, in the sense that it allows the element databases to run to completion and then certifies the global ordering. In particular, the O-vector is constructed only after the subtransactions have finished. Since some concurrency control

techniques (such as time-interval based and optimistic) decide the transaction ordering only at the transaction commit time, it is difficult for the superdatabase to impose an ordering during subtransaction execution. In other words, the superdatabase has to be as optimistic as its element databases.

2.4 Run-Time Cost

In the element databases, the cost of producing the O-vectors is low. First, with some concurrency control methods, such as timestamps, this is trivial. Second, for centralized element databases, taking a timestamp is cheap. Third, only if the element database is a distributed database with internal concurrency control a global clock will be necessary to capture the serial order. Fortunately, the maintenance of a global clock is independent of the number of transactions, and therefore its cost can be amortized.

On the superdatabase side, the certification of an O-vector implies the comparison with all committed supertransactions, which is potentially expensive both in terms of storage and processing. Fortunately, it is not necessary to compare the O-vector with all committed supertransactions. It is sufficient to certify the transaction with a reasonably "recent history" of committed supertransactions.

The part of the serialization history we have to look at is limited by the oldest active transaction in each element database. Suppose we are certifying an O-vector whose subtransactions are older than the currently oldest active transaction on all element databases. Comparing this O-vector to the history of all committed supertransactions, we may not be able to certify this O-vector because of some other older transaction, in which case it must be aborted. Alternatively, we may find a place in the serialization history for the O-vector. Once we find such an O-vector(T_0) preceding all active subtransactions, it must precede the O-vectors of all serializable supertransactions that have yet to commit. This happens because any subsequent O-vector must have one component preceded by the corresponding component in T_0. (The component that was active when T_0 was certified.) Consequently, either the new O-vector cannot be serialzed with respect to T_0 and is aborted, or all its components are preceded by T_0. Therefore, in the certification process we need only to compare the new O-vector with T_0 and O-vectors more recent than T_0. Thus the O-vectors preceding T_0 are not necessary and can be discarded.

For each supertransaction, the only piece of information that the superdatabase needs from the element databases is the O-element. Since an agreement protocol is necessary for recovery purposes, at least one round of messages must be exchanged between the superdatabase and each element database at commit time. The certification occurs only at commit time.

so the subtransaction serial order information can piggyback on the commit vote message. Therefore, the hierarchical superdatabase does not introduce any extra message overhead for heterogeneous transaction processing (compared to homogeneous distributed transactions).

3 Optimization and Distribution

3.1 Hierarchy Flattening

Hierarchical algorithms (e.g. hierarchical two-phase commit) have run-time dominated by the depth of the tree. One standard optimization technique on hierarchical algorithms to reduce response time is to flatten the hierarchy by coalescing all internal nodes into one root node, which communicates to all leaf nodes directly. The flattened tree uses the same algorithm (simplified to one level), with run-time bounded only by the slowest link.

Since flattening occurs at the time a new node joins the tree, its implementation is straightforward. The superdatabase maintains a list of element databases attached to it, with their attributes including the local concurrency control and commit agreement protocol used. Instead of creating a hierarchy, we attach all element databases under the same root superdatabase. This optimization does not change the internal structure of element databases. A hierarchical element database like R* will maintain its own tree structure.

Flattening simplifies the structure of the whole system, but also accentuates the centralization problem. Only one root superdatabase may become both a bottleneck for performance and Achilles' heel for availability. We will take advantage of the simpler structure in section 3.2 to increase transaction concurrency and address the centralization problem in section 3.3.

3.2 Concurrency Control Grouping

O-vectors in hierarchical certification reflect a total ordering guaranteed by serializability. Since some concurrency control methods (e.g. two-phase locking and optimistic) use partial orderings, O-vectors do not capture all the ordering information. Consequently, hierarchical certification may err on the conservative side, aborting some transactions that appear non-serializable because of the particular total ordering, even though they could be serialized in the partial ordering. To preserve the degree of concurrency allowed by each concurrency control method, we apply our knowledge of each particular method.

First, element databases using strict two-phase locking do not have to be certified against each other. Since their lock points are synchronized by the hierarchical commit protocol, they are serialized with respect to each other. In contrast, the example in section 2.3.1 shows that the general two-phase locking requires more care. We need to synchronize the lock

point of element databases through an agreement protocol. Once all subtransactions agree they have reached the global lock point, they can start unlocking, but not before. This way, the superdatabase maintains global two-phase locking and serializability. This conceptual simplicity hides some implementation complications. Even though each element database already supports an agreement protocol for transaction commit, to use the protocol for lock point synchronization requires additional changes on the element databases.

Second, some concurrency control methods such as optimistic certification maintain explicit transaction serialization information such as dependency graphs. The superdatabase can also use that information. For example, if each supertransaction commit message from the element database carries the transaction id of all the subtransactions it depends on, the superdatabase can construct the global dependency graph and use it to certify optimistic transactions. Again, the implementation requires potential extra messages for a supertransaction with many subtranctions and the maintenance of global dependency graph in the superdatabase.

Third, timestamp-based element databases could provide the superdatabase with additional information. For example, time-interval based concurrency control methods would allow the superdatabase to serialize some transactions that would have been aborted in the hierarchical superdatabase. In this case, the implementation is a relatively straightforward transformation of existing timestamp-based concurrency control to time-intervals.

In summary, we can avoid or refine the certification within each group of element databases using the same concurrency control method. However, a global certification must be carried out between different groups. In this higer level certification, each group participates with one O-element. Therefore, supertransations aborted due to non-serializability necessarily come from different groups.

3.3 Symmetric Distribution

As we have seen in section 2, hierarchical organization of superdatabases results in low message overhead. However, the main disadvantage of the hierarchical structure is its centralized organization. Shutting down any of the internal nodes will isolate parts of the tree. In the optimized version, the root superdatabase appears to be a single point of failure that can make the whole heterogeneous system inaccessible.

In reality, the two functions of the root superdatabase, concurrency control and crash recovery, can be independently distributed for parallelism and availability. For recovery, the coordinator of two-phase commit protocol for different supertransactions may be located in different physical nodes. There are two requirements for this distribution scheme. First, the

coordinator node should know how to translate between different protocols that subtransactions use. Second, the coordinator should know how to communicate with concurrency control to make sure the supertransaction is serializable before committing it. Since any superdatabase replica would satisfy these requirements, each can coordinate the commit protocol.

The situation is more complicated for concurrency control. We could replicate the global certification information in superdatabase nodes, resulting in higher message overhead to keep the replicas consistent. Simple replication comes close to being the "brute force" method to distributed functions in a distributed system. In principle, just about any program or data can be distributed this way, provided that they are kept consistent. Unfortunately, consistent replication is expensive and this approach then loses the low-overhead advantage of hierarchical superdatabase.

Alternatively, we can circulate the concurrency control certification information among several sites. This approach is similar to the work by Ceri and Owicki [6] in distributing the optimistic concurrency control certification algorithm. Again, higher message overhead will be necessary. A reasonable compromise would be a central root superdatabase for normal certification. Periodic checkpoints will send the global wait-for-graph to backup sites. If the root node crashes, one of the backups will take over to try to reconstruct the situation before the crash. The trade-offs between normal processing cost and reocvery time are similar to other distributed systems.

4 Implementation

In Fall 1987 We have started a prototype implementation of the superdatabase architecture, called Supernova. Currently, the author, two Ph.D. students and twelve M.S. project students are involved in the implementation effort.

Under Supernova, we have three element databases to ensure heterogeneity. First, we are modifying the university version of INGRES by adding two-phase commit protocol and O-vectors. The university INGRES runs on Berkeley UNIX. For simplicity, we have left the local recovery algorithm "as is" and therefore incorrect. Since INGRESS/STAR already supports two-phase commit with appropriate local recovery, we do not have a point to prove.

Second, we are experimenting with CAMELOT [27], a distributed transaction library package being developed at CMU, which runs on the MACH operating system [1]. We plan to write a CAMELOT server as an element database. Since MACH is binary-compatible with Berkeley UNIX, we expect to run both university INGRES and CAMELOT on MACH. Thus,

the first version of the superdatabase prototype will run on MACH connecting the modified INGRES element database and the CAMELOT element database.

Third, we are writing a relational database of our own design, called Nova, using the Synthesis operating system [22]. Nova consists of three parts: an SQL query compiler and execution manager, a two-phase locking concurrency control manager, and a log-based recovery manager. Currently, all modules are being written in C on UNIX with code synthesis and portability to Synthesis in mind.

The second version of the superdatabase will run on Synthesis. The element databases will run on a Berkeley UNIX emulator and a MACH emulator under Synthesis. To achieve real integration, we still need to extend our query compiler to translate supertransactions into QUEL for the INGRES element database. Another significant technical aspect is the integration of schemas from different element databases, which is not the focus of our research but remains an important practical issue.

The final observation on the implementation of the superdatabase is that we do not plan to modify the code of commercial databases to make them composable element databases. Today, hardware manufacturers publish machine specifications so people in universities and independent software houses can write operating systems for them. Analogously we expect software houses to publish the interface to basic software so we can write higher level software and applications. This practice is prevalent in the PC market, since the added value of third-party software justifies the open system approach. Once Supernova is running, we expect to integrate new element databases through their interface specifications. The implementation details of commercial element databases can remain proprietary.

5 Related Work

5.1 Crash Recovery

Gligor and Luckenbaugh [10] have discussed the recovery problem in heterogeneous databases, without describing specific algorithms. The hierarchical commit algorithm described in section 2.2.1 is a direct descendent of distributed commit protocols such as R* [16] and commit protocols for nested transactions [23]. Each commit protocol known by the superdatabase is exactly the same as in homogeneous distributed databases. The superdatabase only needs to translate between different protocols.

5.2 Concurrency Control

Gligor and Popescu-Zeletin [11] studied concurrency control in heterogeneous databases. They specified five conditions which should be satisfied by concurrency control mechanisms for heterogeneous databases. First, all local concurrency control (of component databases) must provide local synchronization atomicity. We also make this assumption. Second, all local concurrency control must preserve the relative order of execution determined by the global transaction manager. This corresponds to a pessimistic approach, in contrast to the superdatabase's certification after transaction execution. Their third condition says that each site can run only one subtransaction. The superdatabase makes a similar assumption, that each element database runs only one subtransaction. Their fourth condition says that the global transaction manager must be able to identify objects referenced by all subtransactions. Using explicit serialization order in O-elements, we have eliminated the need to check object references. Finally, their fifth condition refers to global deadlock detection. Deadlocks remain a problem for further research.

Another approach in heterogeneous concurrency control assumes existing centralized databases that cannot be modified. Breitbart el al. [5], and Elmagarmid and Helal [7] have studied algorithms under this assumption. The main advantage of this approach is that research results can be applied immediately to existing databases. The main problem of Elmagarmid and Helal's work is the restriction on the class of serializable global transactions: transactions that read and write to more than one site, such as typical fund transfer transactions, cannot be allowed. Breitbart et al. propose the notion of site graph to guarantee global consistency. Site graphs limit transaction concurrency in a different way: their multidatabase may run one transfer transaction but not two concurrently between the same databases.

5.3 Partial Integration

In contrast to our "strongly consistent" database composition, significant work has been done based on weaker consistency constraints. Two examples of this approach are MRDSM [18] and ADMS± [26]. Being developed at INRIA, the prototype multidatabase system MRDSM provides a relational interface to independent databases. Instead of global schemas, special "dependency schemas" define interdatabase relationships. No consistent updates are included in MRDSM.

ADMS± takes advantage of current hardware advances to integrate a mainframe database (ADMS+) with workstation databases (ADMS−) downloaded from the mainframe. Since

each user typically uses only a portion of the database, local queries on ADMS− data are very efficient. Updates occur only on ADMS+ and they are incrementally propagated to ADMS− databases offline. In summary, ADMS± can be seen as a systematic decomposition of a centralized database.

5.4 Standardization

Another way to solve the heterogeneity problem is to decree homogeneity through a standard. However, there are several difficulties in the adoption of a standard. First, political difficulties may arise from financial and other interests; for example, hardware and software vendors may disagree on issues such as market share and competitive advantage; another example is that governments may be unable to agree due to local rules and regulations. Second, standards necessarily contain compromises and may exhibit the symptoms of "design by committee". Third, the difficulties that take a long time to reach agreement may make the standards technologically obsolete by their publication.

There is a working group drafting an ISO standard on distributed transaction processing [4]. The most visible result is the informal agreement on a variant of two-phase commit as the standard commit protocol. One technical reason the superdatabase will be useful both before and after the adoption of the ISO standard is that concurrency control remains an open issue, outside of the standard.

In addition, there are several fundamental reasons the superdatabase complements standards. First, the implementation of the superdatabase depends only on technical information, not political or marketing compromises. Second, the superdabase can handle concurrency control and crash recovery that do conform to different standards, or no standard at all; this is especially important for the integration of new technologies (e.g. long transactions) and their applications (e.g. CAD/CAM/CASE databases). Third, during the transition period from one technology to another both the old and the new need to be running side by side: this requires the degree of integration beyond adherence to a standard.

6 Conclusion

Compared to the traditional computer system heterogeneity problem, heterogeneous updates across databases is more subtle. At the basic level is the translation between different data representations, mapping bit patterns into other bit patterns. A more complicated problem is the translation between different communications protocols, mapping finite state machines into other finite state machines. Unlike data and protocol translations, which map one state

representation into another, atomic transactions across heterogeneous databases require the cooperation between different algorithms, with a specific difficulty in concurrency control. We call this an instance of the *algorithmic heterogeneity* problem.

We propose the superdatabase architecture to support atomic transactions across heterogeneous databases. Starting with hierarchical composition of heterogeneous concurrency control and crash recovery methods, followed by step-wise refinements to reduce run-time overhead and increase transaction concurrency, we have described a family of superdatabase designs with different trade-offs in terms of flexibility, performance, and simplicity.

Several reasons make the superdatabase an attractive approach to consistent heterogeneous databases. First, superdatabase structure is straightforward and we are completing a prototype implementation. Second, superdatabase performance should be good. No transaction concurrency is lost for element databases that share the same concurrency control method. Run-time overhead in both CPU and messages is low. Third, replication and distribution of superdatabase for parallelism and availability is easy, although distribution of concurrency control will carry additional message overhead.

Given the inherent difficulties in the process of agreeing on standards, superdatabases constitute an attrative technical solution to integrate heterogeneous databases. Instead of a rigid standard, superdatabases can bridge the gap between different standards. This will be useful in making the transition from an older standard to an emerging standard due to new technology. Concretely, to integrate commercial databases, we need only to know the interface specification to their concurrency control and crash recovery mechanisms. Currently, the Nova element database and the Supernova superdatabase prototype are being completed at Columbia University.

7 Acknowledgement

I would like to thank Phil Bernstein and Ahmed Elmagarmid for stimulating discussions. Also, I would like to thank all the graduate students actively involved in the implementation of the superdatabase, Ariel Blumencwejg, Mohamed Choudhry, Edward Hee, Masato Inada, Heidi Jones, Paul Kanevsky, Wen-Hwei Lin, William Lin, Pierre Nicoli, Michael Sokolsky, Eugene Temple, Magdeline Vargas, Holger Veith, with special recognition to Avraham Leff, Surasak Lertpongwipusana and Shu-Wie Chen.

References

[1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young.

Mach: a new kernel foundation for Unix development.
In *Proceedings of the 1986 Usenix Conference*, pages 93–112, Usenix Association, 1986.

[2] R. Bayer, K. Elhardt, J. Heigert, and A. Reiser.
Dynamic timestamp allocation for transactions in database systems.
In H. J. Schneider, editor, *Distributed Data Bases*, North-Holland, 1982.

[3] P.A. Bernstein, V. Hadzilacos, and N. Goodman.
Concurrency Control and Recovery in Database Systems.
Addison-Wesley Publishing Company, first edition, 1987.

[4] M. Bever, M. Feldhoffer, and S. Pappe.
OSI services for transaction processing.
In *Proceedings of the Second International Workshop on High Performance Transaction Systems*, pages 8.1–8.17, Asilomar, California, September 1987.

[5] Y. Breitbart and A. Silberschatz.
Multidatabase update issues.
In *Proceedings of 1988 SIGMOD International Conference on Management of Data*, pages 135–142, May 1988.

[6] S. Ceri and S. Owicki.
On the use of optimistic methods for concurrency control in distributed databases.
In *Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Networks*, pages 117–129, Lawrence Berkeley Laboratory, University of California, Berkeley, February 1982.

[7] A. Elmagarmid and A.A. Helal.
Supporting updates in heterogeneous distributed database systems.
In *Proceedings of the Fourth International Conference on Data Engineering*, pages 564–571, Los Angeles, February 1988.

[8] K.P. Eswaran, J.N. Gray, R.A. Lorie, and I.L. Traiger.
The notions of consistency and predicate locks in a database system.
Communications of ACM, 19(11):624–633, November 1976.

[9] A. Ferrier and C. Stangret.
Heterogeneity in the distributed database management system SIRIUS-DELTA.
In *Proceedings of the Eighth International Conference on Very Large Data Bases*, Mexico City, September 1983.

[10] V. Gligor and G.L. Luckenbaugh.

Interconnecting heterogeneneous database management systems.
Computer, 17(1):33–43, January 1984.

[11] V. Gligor and R. Popescu-Zeletin.
Concurrency control issues in distributed heterogeneous database management systems.
In F.A. Schreiber and W. Litwin, editors, *Distributed Data Sharing Systems*, pages 43–56, North Holland Publishing Company, 1985.
Proceedings of the International Symposium on Distributed Data Sharing Systems.

[12] H. T. Kung and John T. Robinson.
On optimistic methods for concurrency control.
Transactions on Database Systems, 6(2):213–226, June 1981.

[13] L. Lamport.
Time, clocks and ordering of events in a distributed system.
Communications of ACM, 21(7):558–565, July 1978.

[14] T. Landers and R.L. Rosenberg.
An overview of MULTIBASE.
In H.J. Schneider, editor, *Distributed Data Bases*, North Holland Publishing Company, September 1982.
Proceedings of the Second International Symposium on Distributed Data Bases.

[15] P.J. Leu and B. Bhargava.
Multidimensional timestamp protocols for concurrency control.
IEEE Transactions on Software Engineering, SE-13(12):1238–1253, December 1987.

[16] B. Lindsay, L.M. Haas, C. Mohan, P.F. Wilms, and R.A. Yost.
Computation and communication in R*: a distributed database manager.
ACM Transactions on Computer Systems, 2(1):24–38, February 1984.

[17] B.H. Liskov and R.W. Scheifler.
Guardians and Actions: linguistic support for robust, distributed programs.
In *Proceedings of the Ninth Annual Symposium on Principles of Programming Languages*, pages 7–19, January 1982.

[18] W. Litwin and A. Abdellatif.
Multidatabase interoperability.
Computer, 19(12):10–18, December 1986.

[19] R. McCord.
INGRES/STAR: a distributed heterogeneous relational DBMS.

Vendor Presentation in SIGMOD, May 1987.

[20] J.E.B. Moss.
Nested Transactions: An Approach to Reliable Distributed Computing.
PhD thesis, Massachusetts Institute of Technology, April 1981.

[21] C. Pu.
From nested transactions to supertransactions.
1987.
Submitted for publication.

[22] C. Pu, H. Massalin, and J. Ioannidis.
The Synthesis kernel.
Computing Systems, 1(1):11–32, Winter 1988.

[23] Calton Pu.
Replication and Nested Transactions in the Eden Distributed System.
PhD thesis, Department of Computer Science, University of Washington, 1986.

[24] Calton Pu.
Superdatabase for composition of heterogeneous databases.
In *Proceedings of Fourth International Conference on Data Engineering*, pages 548–555, IEEE/Computer Society, Los Angeles, February 1988.

[25] D.P. Reed.
Naming and Synchronization in a Decentralized Computer System.
PhD thesis, Massachusetts Institute of Technology, September 1978.

[26] N. Roussopoulos and H. Kang.
Principles and techniques in the design of ADMS±.
Computer, 19(12):19–25, December 1986.

[27] A.Z. Spector, D.S. Thompson, R.F. Pausch, Eppinger J.L., D. Duchamp, R.P. Draves, D.S. Daniels, and J.J. Bloch.
Camelot: A Distributed Transaction Facility for Mach and the Internet - An Interim Report.
Technical Report CMU-CS-87-129, Computer Science Department, Carnegie-Mellon University, June 1987.

[28] M. Templeton, D. Brill, S. K. Dao, E. Lund, P. Ward, Chen A.L.P., and R. MacGregor.
MERMAID — a frontend to distributed heterogeneous databases.
Proceedings of the IEEE, 75(5):695–708, May 1987.