Implementation of a Prototype Superdatabase

Calton Pu and Shu-Wie F Chen
Department of Computer Science
Columbia University
New York, NY 10027

1 Introduction

Since 1986 we have been developing the Harmony heterogeneous database system in the Department of Computer Science at Columbia University. The major focus of the Harmony project has been heterogeneous transaction processing (HTP), in particular, the superdatabase architecture [7]. This position paper reports on the implementation experience of the Harmony prototype, including the Supernova, an instance of superdatabase, and three different component databases.

We have three goals in the implementation of Harmony prototype. First, we want to test and refine the ideas in the superdatabase architecture. Only through an implementation can we realistically study the validity and applicability of the several design choices. Second, we need to show the practicality of the superdatabase, which has not been completely accepted as an "easy to do" HTP. Third, we are designing a next-generation heterogeneous database in Harmony, and we would like to acquire some solid database system implementation experience.

We believe that all three goals have been achieved to some degree. Section 2 summarizes the superdatabase architecture for completeness. Section 3 describes Supernova, which materializes a superdatabase, and the three component databases. Section 4 concludes with an evaluation of the implementation and immediate applications.

2 The Superdatabase Architecture

2.1 Composition
The superdatabase architecture provides distributed atomic transactions across heterogeneous databases. A superdatabase is conceptually a hierarchical composition of component databases, which may be centralized, distributed, or other superdatabases. Transactions spanning multiple component databases are called supertransactions. Each local transaction is a subtransaction. We say that a component database is composable if it satisfies the following composability conditions:

- It provides a distributed commit protocol, e.g. two-phase commit.
- It guarantees recovery atomicity for local transactions, e.g. logging.
- It guarantees serializability for local transactions, e.g. two-phase locking.

These conditions are mild, since most homogeneous distributed databases satisfy them. Concrete examples of composable component databases include the announcements of commercial databases such as Oracle and INGRES.

The superdatabase has two main parts: crash recovery and concurrency control. We assume that the readers are familiar with the transaction processing literature [1] and will only summarize the ideas and algorithms of superdatabase for HTP.

2.2 Heterogeneous Crash Recovery

Traditional distributed (homogeneous) databases use some kind of agreement protocol to ensure that all of the subtransactions either commit or abort together. (The only exceptions are systems that provide asynchronous operations.) Here, we assume that all component databases provide some form of agreement. For example, in R* [6], two-phase commit implements hierarchical commit.

The question is whether the agreement is sufficient for heterogeneous commit. The answer is yes.
Commit protocols such as two-phase commit obtain agreement on the outcome of the transaction independently of recovery information to undo/redo updates. Therefore, provided the superdatabase understands and uses the appropriate protocol, we do not have to deal with the subtransaction undo or redo.

Since the superdatabase is the coordinator for the component databases during commit, it must write the commit record to the log. Conceptually, the superdatabase log is separate from the component database logs, just as the superdatabase itself is separate from the component databases. In actual implementation, the superdatabase log may be physically interleaved with a component database log, as long as the recovery algorithms can separate them later. For each supertransaction, the superdatabase saves enough information for recovery in the case of crashes. Our implementation uses a separate log.

2.3 Heterogeneous Concurrency Control

Different heterogeneous concurrency control mechanisms introduce difficult problems for HTP. In general, subtransactions from different component databases may be serialized in different ways, so even if each subtransaction has been serialized locally, the supertransaction is globally non-serializable. For example, consider transactions $T_1$ and $T_2$ running on the same component databases $A$ and $B$. It is possible that the subtransactions on $A$ have been serialized one way and on $B$ the other way, for example, $T_{1,A} \leq T_{2,A}$ and $T_{2,B} \leq T_{1,B}$. This scenario is quite plausible when $A$ uses a concurrency control that serializes at the beginning of transactions, say basic timestamps, and $B$ serializes at the end, say optimistic validation.

To prevent this kind of inconsistency, we define an order-element (O-element) as the serialization order of each subtransaction in a component database: $T_{1,A} \leq T_{2,A}$ if and only if $T_{1,A}$ has been serialized before $T_{2,A}$. An order-vector (O-vector) is the concatenation of all O-elements of the supertransaction. In the above example, O-vector($T_1$) is (O-element($T_{1,A}$), O-element($T_{1,B}$)). The order induced on O-vectors by the O-elements is defined strictly: O-vector($T_1$) $\leq$ O-vector($T_2$) if and only if for all component databases $j$, O-element($T_{1,j}$) $\leq$ O-element($T_{2,j}$). If a supertransaction is not running on all component databases, we use a wildcard O-element, denoted by * (star), to fill in for the missing component databases. By definition, O-element(any) $\leq$ *, and, * $\leq$ O-element(any).

This definition implies that if O-vector($T_1$) $\leq$ O-vector($T_2$) then all subtransactions are serialized in the same order. Therefore, by checking the O-vector of a committing supertransaction against the history of all committed supertransactions we can serialize the supertransactions. If the new O-vector finds a place in the total order, it may commit. If the concurrency control methods do not allow update transactions to commit in the past, then we only have to compare the currently committing supertransaction with the recent history. Otherwise, for example in optimistic validation, we have to either maintain a longer log or to give the dependency explicitly to the local database. Either technique will preserve the global serializability for the rare case of serialization in the past.

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 component databases. As long as we can make the serialization order in component databases explicit, the superdatabase can certify the serializability of supertransactions. Therefore, a superdatabase can compose two-phase locking, timestamps, and optimistic concurrency control methods. In addition, the certification gives the superdatabase itself an explicit serial order (the O-vector) allowing it to be recursively composed as a component database.

The certification method is optimistic, since it allows the component databases to run to completion and then certifies the serial ordering. In particular, the O-vector is constructed only after the subtransactions attempt to commit. Since some concurrency control techniques (such as time-interval based and optimistic) decide the transaction ordering only at the transaction commit time, this is a natural choice. In other words, the superdatabase is as optimistic as its component databases.

2.4 Optimization and Distribution

Although we have not implemented in Supernova the optimizations possible in the superdatabase architecture, we outline them here for completeness. The hierarchical composition of component
databases can be followed by optimization to increase performance and decrease concurrency loss. Some of the optimization techniques are well-known, for instance, the flattening of hierarchical structures to decrease communications costs in the hierarchical two-phase commit protocol.

Other optimization techniques are specific to the composition of databases. To increase concurrency, we group together database managers that use the same concurrency control method, say two-phase locking, and exploit that knowledge. In particular, two-phase locking databases achieve global serialization if we synchronize the local lock points, such as in strict two-phase locking. To eliminate the root bottleneck, we can distribute the commit coordinator and global serializability validation. Additional details on the superdatabase architecture can be found in the reference [7].

3 The Prototype

3.1 Historical Perspective

The implementation effort started in Fall of 1987, when about 15 students took the Advanced Database course. The students were divided into groups to do a practical project. We divided the projects into three layers of software: OS support, Component DB, and Superdatabase. The OS support group worked on a common interface intended for higher portability and performance of databases. The Superdatabase group designed a simple instance of Superdatabase and subsequently implemented it (called Supernova). The Component DB layer was further divided into four groups: storage manager, concurrency controller, query compiler, and data dictionary. This layer became the first version of the Nova relational database.

Since then, more than 20 project students have worked on the Harmony project (for a partial listing see section 5). Averaging about 2.5 project-semesters each, the total implementation effort exceeds 50 project-semesters. All of the major components have been re-implemented at least once. All the code is written in C on some flavor of UNIX. The current working code adds up to about 30,000 lines. From the management point of view, the stability provided by the core group (the authors) has been instrumental in keeping the project together.

3.2 Supernova

The Supernova glues the system together. It distributes the global supertransaction to the component databases, logs the supertransaction for recovery, and validates the global serialization of the component transactions when they complete. The current version of Supernova does not include the optimizations described in section 2.4 but implements full recovery and concurrency control summarized in sections 2.2 and 2.3. It runs under Ultrix on a Microvax and is referred to as Supernova/Ultrix.

Supernova has two important parts. First, it maintains global concurrency control. The concurrency control module maintains a list of O-elements for each component database. Each item in the list represents the O-element returned by a component database participating in a supertransaction. Currently, we have two families of concurrency control techniques among the component databases: two-phase locking and optimistic validation (see section 3.3). For the two-phase locking, the O-element comes from the timestamp of the lock point of each local transaction. The optimistic validation produces its O-elements directly.

The second important part of Supernova is the distributed commit. The key problem in heterogeneous commit is the translation between different commit protocols. Supernova maintains a table of procedures that implements the commit process for each type of commit protocol. The actual commit protocol is table-driven to allow easy addition of new protocols. The component databases described below all use the Nova protocol, since we had to add the protocol to them. The inclusion of LU6.2 is in progress.

The total effort spent on Supernova has been non-trivial. Currently we have about 7000 lines of code in the working version. Much of it has been re-implemented, so the total amount of code written far exceeds the current version.

3.3 Component Databases

Supernova/Ultrix currently integrates three different component databases: a modified version of university INGRES running on SUNOS, a CAMELOT server running on the MACH operating system (on a Microvax), and our own Nova relational database manager. Each of the component databases was
built differently. Many project students wrote Nova-2PL from scratch. Nova-2PL includes an SQL/C query compiler, rudimentary data dictionary support, two-phase locking concurrency control, and simple recovery mechanisms using the UNIX file system. The current version runs on Ultrix. This is our standard “learning” database system since it is modular. The Nova-2PL concurrency controller is about 8000 lines, the data manager 4000 lines, and query compiler (without query optimization) 1500 lines.

One of the most recent modifications is the addition of more concurrency heterogeneity with Nova-OCC, which substitutes optimistic validation for two-phase locking in Nova-2PL. The other modules of Nova-OCC are shared with Nova-2PL. The optimistic concurrency controller is about 5000 lines and took about three project-semesters.

The university version of INGRES is a good example of a centralized database made composable. We added the two-phase commit protocol, made the recovery routine understand the “prepared” state, and returned the order-vector to the Supernova. The conversion of the centralized INGRES to a composable INGRES took about one man-year total of two very competent MS project students. The total number of lines changed was about 2000. The availability of a commercial INGRES satisfying the composable conditions was not a surprise for us.

The CAMELOT server is called Jake, a slightly modified version of the Jack server distributed with the CAMELOT package. Jack executes simple transactions and returns results. Jake converts the CAMELOT commit protocol to the Supernova protocol. Although the CAMELOT servers by design satisfy the composability conditions, we had some difficulties with the interface. We gladly acknowledge the help from Prof. Dan Duchamp of Columbia University, who wrote the original CAMELOT transaction manager. The current working version of Jake is about 900 lines of code, which has been rewritten three times.

4 Evaluation and Conclusion

Some of the workers in the heterogeneous systems field believe that most of the work should go into the establishment of a standard. Once we can agree on a standard, the entire heterogeneity problem will disappear by decree. This would be true if we could agree on standards quickly and stay with them for a long time. As it happens, standards take a long time and many discussions to mature. By the time a standard is established, new technology already suggests better solutions for the problem at hand.

We propose the superdatabase architecture as a solution for heterogeneity before, during, and after standards. Moreover, the superdatabase is useful for bridging different standards such as LU6.2 and ISO/TP. Finally, as new technology arises, we can move on and experiment with them under the superdatabase, until we reach the next generation of standards.

The current implementation effort of Supernova focuses on the interfaces and portability. First, we want to port the Supernova/Ultrix to run on SUNOS and HP-UX, so more people can use it. Second, we have started the implementation of Supernova protocol support for LU6.2 and we are studying the ISO/TP. In this way, we will be able to integrate open system databases following different standards.

The total implementation effort of Harmony prototype is similar in magnitude to several of the prototypes reported in a recent special issue of IEEE Transactions on Knowledge and Data Engineering [8]. The degree of heterogeneity, global atomicity, amount of concurrency preserved, and the low overhead of Supernova compares favorably with the HTPs reported in the literature [3, 5]. This result is relevant given the relatively modest implementations reported in a recent special issue of Computing Surveys on Heterogeneous Databases [4]. The details of the implementation will be reported in another paper [2].

5 Acknowledgment

Many project students contributed to the implementation of Harmony prototype. In alphabetical order, undergraduate project students: Jeff Alvidrez, Steven Harari; MS project students: Ariel Blumencwejg, Heidi Jones, Suresak Lertpongwisana, Pierre Nicoli, Mike Sokolsky, Vanessa Sun, Nathan Tanuwidjaja, Les Temple, Boris Umylny, Magdaline Vargas, Holger Veith, Albert Wang; PhD student: David Fox. At least seven others made minor contributions.
This work is partially funded by the New York State Center for Advanced Technology on Computer and Information Systems under the grant NYSSTF CU-0112580, the National Science Foundation under the grant CDA-88-20754, the AT&T Foundation under the Special Purpose Grant program, the Digital Equipment Corporation under the External Research Program, and IBM Fellowships.

References

Concurrency Control and Recovery in Database Systems. 

Supernova: an implementation of the superdatabase. 

Special Issue on Distributed Database Systems, volume 75:5 of Proceedings of the IEEE. 

Special Issue on Heterogeneous Databases, volume 22:3 of ACM Computing Surveys. 
ACM, September 1990.

Integration of Information Systems: Bridging Heterogeneous Databases. 

A retrospective of R*: A distributed database management system. 

Superdatabases for composition of heterogeneous databases. 

Special Issue on Database Prototype Systems, volume 2:1 of IEEE Transactions on Knowledge and Data Engineering. 