StarBase Research Summary
Autumn 1994

Matthew R. Lehr and Sang H. Son
mrl6a@cs.Virginia.EDU
son@cs.Virginia.EDU

Department of Computer Science
School of Engineering and Applied Science
University of Virginia
Charlottesville, VA 22903, USA

Abstract

This report describes the research activities pertaining to the StarBase project at the University of Virginia's Department of Computer Science. StarBase, a firm real-time database management system (RT-DBMS), is remarkable in that it is constructed on top of an actual real-time microkernel. StarBase seeks to provide a general RT-DBMS environment which can handle a variable transaction load, rather than statically scheduling a known set of transactions. This report presents an overview of the StarBase RT-DBMS as it operates today and describes its development environment. The accomplishments of the last two years of research are reviewed and the critical features needed to support such a RT-DBMS are examined.

1. Introduction

The aim of the RT-DBMS group[1] at the University of Virginia is to construct a database server for time-critical applications. Previous academic work in the area of real-time database technology has primarily focused on transaction scheduling and implementations have been exclusively simulations. Commercial RT-DBMS are typically limited to fixed scheduling of transactions determined off-line and tailored to a specific application. StarBase has taken a different tack: it has sought to apply existing real-time techniques such as priority-based scheduling and

1. At the time of this writing, a general description of the group's research activities was available via the World-Wide Web: `http://www.cs.virginia.edu/~vadb/`

resource preallocation to transform a simple relational DBMS into a flexible, firm deadline-enforcing RT-DBMS capable of servicing a variable transaction load. In order to provide the maximum degree of predictability, StarBase has been constructed on top of Real-Time Mach (RT-Mach) [Tok90]. RT-Mach is an experimental real-time microkernel under development at Carnegie Mellon University. RT-Mach provides a real-time thread model [Tok90], where threads have explicit criticalities and timing constraints, priority-based, time-driven CPU scheduling, time- and priority-based synchronization [Tok91], priority-based interprocess communication (IPC) [Kit93] with priority handoff, and processor reservation facilities. StarBase uses RT-Mach's real-time features as a basis for real-time transaction processing and augments the DBMS with special priority-cognizant concurrency control, a more precise conflict detection scheme, and deadline handling.

This report briefly outlines StarBase's major features and goes on to describe the RT-DBMS group's development environment and research accomplishments to date. Next, the group's analysis of the essential features an RTOS must have in order to support a RT-DBMS are presented, followed by a discussion of directions for future research.

2. Major Features

2.1 Conventional Features

- relational model
- simple, SQL-like query language
- concurrent transaction execution
- serializability as correctness criterion
- client/server model

2.2 Real-Time Features

- transactions have explicit criticalities, timing constraints
- priority-driven transaction processing
- WAIT-X(S) optimistic, priority-based concurrency control [Har91]
- Precise Serialization conflict detection to reduce aborts [Lee93, Lee94]
- firm deadline enforcement
- eagerly-allocated deadline handler
- priority-mapping scheme ensures deadline handler takes precedence over transaction manager without interfering with higher priority transactions
- race between commit and deadline expiration resolved by concurrency controller
- memory-based relations
- T-Tree indexing [Leh86a, Leh86b], designed especially for memory-based relations
- transaction manager preallocation

3. Development Environment

The RT-DBMS lab is developing the RT-DBMS on the following platforms:

Gateway 2000 486DX2/50E
16 MB memory
Western Digital Caviar 2200 212.6 MB IDE Drive
3Com Etherlink II Ethernet NIC
RT-Mach MK83g[1]

Gateway 2000 4DX2-66V
16 MB memory
Western Digital Caviar 2340 341.2 MB IDE Drive
3Com Etherlink II Ethernet NIC
Alpha Logic STAT! System Timing Board
RT-Mach MK83g

The following platform, on loan from Loral Federal Systems, will eventually serve as either as a target platform for a port of StarBase or it will serve as a host for a commercial RT-DBMS:

i486 Clone
4 MB memory
Hewlett-Packard HP 97560E 1600 MB SCSI Drive
3Com Etherlink II Ethernet NIC
QNX 4.20

4. Accomplishments

This section describes the specifics of the research involved in producing StarBase and gives some insight into the motivation behind many of its features. The StarBase architecture itself is described in [Leh93, Leh94].

4.1 Port to RT-Mach

In the Autumn of 1992, the RT-DBMS received its first release of RT-Mach (MK78) and began the port of what was then called the RTDB real-time DBMS [Kim94] from the ARTS real-time operating system to RT-Mach [Son93]. In anticipation of the port's completion, the group designed a workload generator which, given a transaction description file, spawned threads which submitted the transactions at the designated periodic intervals with the appropriate operations, deadlines, and criticalities to the DBMS server. By the early Spring of 1993, the port to RT-Mach

1. At the time of this writing, the latest release of RT-Mach was available via `ftp://ftp.cs.cmu.edu/afs/cs/project/rtmach-5/ftp/releases/`

was complete and the DBMS was re-christened *StarBase*.

During the Spring of 1993, StarBase's lock-based concurrency control was replaced by the first implementation of an optimistic algorithm, WAIT-X [Har91], developed at the University of Wisconsin. A real-time optimistic concurrency controller has the advantages that it tends to resolve conflicts late in a transaction's execution and that it can take into account a transaction's feasibility when making the decision of whom to abort. T-Tree indexing [Leh86a, Leh86b], developed specifically for memory-resident relations, was also added during this period.

4.2 Refinements

By the Summer of 1993, it had become apparent that StarBase needed significant changes in its ARTS-style architecture. ARTS had limited its processes to eight distinct priority levels, but RT-Mach's real-time thread model allows a more flexible scheme, combining criticality with explicit timing constraints. Because the ARTS priority scheme was so limited, RTDB on ARTS employed a workload mediation scheme to map an incoming request onto the server process which would process it. The problem was that each server process was dedicated to a single priority level under the mediation scheme so requests could go unserviced for long periods even if there were idle server processes.

The remedy was to remove the workload mediation scheme. Server threads in StarBase were made equals, both in their scheduling priorities and in how they received transaction requests: server threads could process a transaction of any priority level. Since RT-IPC wasn't available in RT-Mach MK78, a dispatcher thread first queued incoming requests in priority order before server threads could begin processing them. In this way the next available server thread was guaranteed to service the highest-priority unserviced request. The elimination of workload mediation also eliminated the need to copy the incoming request into the serving thread's workspace.

The unwieldy applications programming interface (API) to the DBMS server was the next to go. The RTDB on ARTS API was cumbersome, requiring the use of several function calls just to execute one database operation. The StarBase API was changed to free the client from port and message management and to convey user-specified priority and timing constraints to the DBMS server. Rather than pass information in large, worst-case-sized data structures, the new API allowed clients to specify operations as text, a much more compact and flexible representation.

Mid-way through the Summer, the concurrency control introduced in the Spring was redesigned, prompting the writing of a technical report [Leh93]. The description of WAIT-X, which seems so simple on paper, actually proved quite difficult to implement. The redesign aimed to be a much more independent module than the preceeding one had been: this time the new implementation had a definite interface and the implementation was kept as separate

as possible from the rest of the DBMS. The original implementation didn't provide enough mutual exclusion to work correctly, so the new concurrency controller, CCMgr, became its own server, providing each operation in the interface as a service. The CCMgr now actively prevented potential conflictors from advancing in their read sets during the critical wait-test. The CCMgr's data structures were redesigned to provide faster validation, and an asynchronous abort mechanism was built in to handle deadline enforcement. The asynchronous abort mechanism was to be called by a lazily-allocated deadline handler, and the CCMgr was expected to resolve the race condition between transaction commit and deadline expiration.

4.3 Concurrency

By the time the Fall of 1993 arrived, it was clear that despite all of the work of the preceeding year, StarBase could still not execute concurrently. StarBase, and RTDB before it, had been derived by adding new features to the SDB simple relational DBMS, which provided a single-user, minimal functionality DBMS. As such, the SDB portion of StarBase was a black box which had never really evolved, despite all of the new features sprouting in the rest of StarBase. A good portion of the Fall was spent trying to understand precisely what was needed to make the SDB-indigenous code work concurrently. All of SDB's global variables were identified; those which were to be shared amongst the server threads were to have mutual exclusion enforced upon them, and those which were really specific to the query at hand were moved into a query-context maintained by each server thread. It turned out that the shared variables pertained to locating relations for opening, closing, and writing relations, so it was decided that the CCMgr could provide mutual exclusion for them to avoid an extra layer of locking. To aid debugging, the newly concurrent StarBase maintained, for the first time, a debugging log in which events could be recorded for later analysis.

By the time Winter recess of 1993-94 had arrived, StarBase had evolved to the point where it could run concurrently for extended periods. It was during this time that the first demonstration program was developed to graphically determine whether or not StarBase was working as advertised. The demo consisted of two bouncing balls each of whose positions was determined by the same set of values stored in a relation at the StarBase server. One of the balls read the values via a low-priority read transaction and the other read the values via a high-priority read transaction. The deadline miss ratios were depicted using thermometer-like graphs below the respective regions where the low- and high-priority balls bounced.

The demo, once debugged fully, definitively showed that StarBase was indeed executing transactions correctly concurrently and that the deadline handling mechanism was working correctly. It also demonstrated that more was needed, besides the priority-based WAIT-X concurrency control, to favor higher-priority over lower-priority transactions. By this time, the ART group had installed RT-Mach MK83b, which fortunately had real-time synchronization (RT-Sync) [Tok91] and message-passing (RT-IPC) [Kit93] facilities. The first weeks of the Spring semester were spent integrating StarBase with these facilities, changing the CCMgr from a server to a monitor, and creating a

Small Memory Manager which was guarded by real-time locks and which wired allocated memory to keep it from paging out. Since RT-Mach features were not yet quite stable and since real-time features are often more time-consuming than conventional features, which to use was left as a compile-time constant.

4.4 Deadline Enforcement

Another major milestone of that Spring was the development of an eagerly-allocated deadline handler. The deadline handler consisted of a real-time thread and a real-time monitor (using facilities from RT-Sync). The real-time monitor allowed the deadline handler and its corresponding transaction manager to synchronize at the beginning and end of the transaction as needed. A real-time condition variable with a time-out allowed the deadline handler to wait for the earliest of two events: its cancellation (if the transaction commits) or its abort of the transaction (if the deadline expires). A special priority mapping scheme to map user priorities onto the priorities at which a transaction manager and its deadline handler actually execute permitted the deadline handler to run at a higher priority than the transaction manager without interfering with higher priority transactions.

4.5 Recent Additions

The final accomplishment of the Spring of 1994 was the implementation of Precise Serialization conflict detection developed at the University of Virginia. The scheme augmented the WAIT-X base concurrency control method, detecting conflicts more accurately and reducing the frequency of conflict aborts.

The focus of the Summer of 1994 was to improve DBMS performance and stability. The SDB parser and expression compiler were radically altered to run faster and SDB was changed to store numbers in two's-complement and floating-point representation rather than as character strings. The T-Tree was redesigned to make more efficient use of memory and to behave correctly when subjected to concurrent operations. StarBase saw many bug fixes and memory leak plugs during this period.

Since that time, the research focus for StarBase has been to identify features which can be preallocated to boost DBMS performance and predictability. Currently, mechanisms to preallocate transaction managers and transaction query contexts are being developed.

5. RTOS Features to Support a Real-Time DBMS

As with any real-time application, RT-DBMS are highly dependent on the underlying RTOS to provide certain functionality. The past two years of StarBase's development have brought a lot of insight into what is needed of any RTOS to support a RT-DBMS:

Transaction scheduling:	Real-time processes/threads which have explicit criticality/timing information Priority-based CPU and resource scheduling to avoid unbounded priority inversion
Soft and firm deadline enforcement:	Time-based synchronization, especially mechanisms which are capable of blocking until a certain point in time or an event occurs, whichever comes first A global real-time clock or reliable real-time communication to enforce deadlines consistently across distinct processors.
Client/server communication:	Priority-based IPC with priority handoff to execute transactions at the proper priority
Memory-resident relations:	Threads/lightweight processes, shared memory synchronization, ability to pin/wire memory
Disk-based relations/database recovery:	Disk hardware and drivers which are integrated with the priority-based CPU and resource scheduling of the host RTOS.

6. Future Work

The research of the past two years has gone a long way in uncovering how to architect a RT-DBMS. StarBase has exploited the real-time features of its host real-time operating system to deal with resource contention and transaction scheduling. StarBase has used special priority-based concurrency control and conflict detection scheme to solve the problem of data contention. An eagerly-allocated, non-interfering deadline handling mechanism has also been developed and exploration of resource preallocation is just beginning.

Once preallocation has been fully explored, the next area of research will be how to produce execution time estimates so that StarBase can better determine transaction feasibility. Off-line and static scheduling of known transaction sets is another area to be tackled. Now that the essential features to support an RT-DBMS have been determined, target RTOS platforms can be identified and StarBase can be ported. Investigation of some practical issues,

such as whether concurrency really does improve performance for main-memory relations and whether WAIT-X really does outperform lock-based real-time concurrency control must be explored.

References

[Har91] Haritsa, Jayant R. "Transaction Scheduling in Firm Real-Time Database Systems." TR1036. Department of Computer Science, University of Wisconsin. Aug 1991.

[Kim94] Kim, Youngkuk, Lehr, Matthew, George, David and Son, Sang H. "A Database Server for Distributed Real-Time Systems: Issues and Experiences." *Second IEEE Workshop on Parallel and Distributed Real-Time Systems*, Cancun, Mexico, Apr 1994.

[Kit93] Kitayama, Takuro, Nakajima, Tatsuo, and Tokuda, Hideyuki. "RT-IPC: An IPC Extension for Real-Time Mach." *Proceedings of the Second Microkernel Workshop*, Sep 1993.

[Lee93] Lee, Juhnyoung and Son, Sang H. "Using Dynamic Adjustment of Serialization Order for Real-Time Database Systems." *14th IEEE Real-Time System Symposium*, Dec 1993.

[Lee94] Lee, Juhnyoung, and Son, Sang H. "Precise Serialization for an Optimistic Concurrency Control Algorithm." Submitted for Publication.

[Leh86a] Lehman, Tobin J. and Carey, Michael J. "Query Processing in Main Memory Database Management Systems." *Proceedings of the ACM SIGMOD Conference*. May 1986.

[Leh86b] Lehman, Tobin J. and Carey, Michael J. "A Study of Index Structures for Main Memory Database Management Systems." *Proceedings of the Twelfth International Conference on Very Large Databases*. Aug 1986.

[Leh93] Lehr, Matthew R. "StarBase v2.2 Implementation Details." TR CS-93-48. Department of Computer Science, University of Virginia. Jul 1993.

[Leh94] Lehr, Matthew R. and Son, Sang H. "Managing Contention and Timing Constraints in a Real-Time Database System." TR CS-94-19. Department of Computer Science, University of Virginia. Jun 1994.

[Sav93] Savage, Stefan, and Tokuda, Hideyuki. "Real-Time Mach Timers: Exporting Time to the User." *Proceedings of the Third USENIX Mach Symposium*, Apr 1993.

[Son93] Son, Sang H., George, David W., and Kim, Young-kuk. "Developing a Database System for Time Critical Applications on RT-Mach." Unpublished.

[Tok90] Tokuda, Hideyuki, Nakajima, Tatsuo, and Rao, Prithvi. "Real-Time Mach: Towards a Predictable Real-Time System." *Proceedings of the First USENIX Mach Workshop*, Oct 1990.

[Tok91] Tokuda, Hideyuki, and Nakajima, Tatsuo. "Evaluation of Real-Time Synchronization in Real-Time Mach." *Proceedings of the Second USENIX Mach Workshop*, Oct 1991.