Fermilab FERMILAB-Conf-03/130 June 2003

RTDB: A Memory Resident Real-Time Object Database

Jerzy M. Nogiec, Eugene Desavouret, *Member, IEEE*

***Abstract*-- RTDB is a fast, memory-resident object database with built-in support for distribution. It constitutes an attractive alternative for architecting real-time solutions with multiple, possibly distributed, processes or agents sharing data. RTDB offers both direct and navigational access to stored objects, with local and remote random access by object identifiers, and immediate direct access via object indices. The database supports transparent access to objects stored in multiple collaborating dispersed databases and includes a built-in cache mechanism that allows for keeping local copies of remote objects, with specifiable invalidation deadlines. Additional features of RTDB include a trigger mechanism on objects that allows for issuing events or activating handlers when objects are accessed or modified and a very fast, attribute based search/query mechanism. The overall architecture and application of RTDB in a control and monitoring system is presented.**

I. Introduction

THERE exist various architectural solutions to the problem of organizing systems consisting of multiple collaborating processes. Let us focus our attention on the exchange of data or objects between processes and consider a solution based on a database. If a set of processes comprising a family of integrated applications all exchange data through the same database then they are guaranteed to be consistent all of the time. Since the purpose of using the database is to organize sharing of data/objects, traditional databases, although offering persistence, are not very attractive for real-time applications because of prolonged access times. Also, persistence for transient or internal data is not typically needed. Instead, one can use a memory-resident database, a database built especially for this purpose that offers fast access times owing to eliminated disk I/O.

Memory-resident databases differ from disk databases in that data resides in the main memory. These databases, working as organized repositories of shared data, are geared toward fast, programmatic access and therefore are especially suited for telecommunications and control applications.

Manuscript received May 9, 2003. This work was supported by the U.S. Department of Energy under Contract No. DE-AC02-76CH03000.

J. M. Nogiec is with the Fermi National Accelerator Laboratory, Batavia, IL 60510 (telephone: 630-840-3081, e-mail: nogiec@fnal.gov).

E. Desavouret is with the Fermi National Accelerator Laboratory, Batavia, IL 60510 (telephone: 630-840-4402, e-mail: desavouret@fnal.gov).

The fast access requirement makes object orientation especially attractive, and favored over a more common relational model.

Several commercial in-memory databases exist, such as eXtremeDB [1], BoostEngine [2], Polyhedra [3], Powerdata SDK [4], or Vaccess [5].

II. Architecture

The Real-Time Database (RTDB) is built as a toolkit. It allows a real-time developer to built an integrated solution encapsulating an in-memory database suitable for his or her application. Only needed features are used and algorithms can be optimized for a given situation.

RTDB is used to keep complex data structures or object states accessible by multiple processes. It resides in a shared memory and is directly accessible by any local process, which allows for a very rapid access to data. Processes can work directly with data, making copying of data unnecessary.

Objects inside the database are accessible via their names, identifiers, and references. Initially access has to be done via a name, and then the program can navigate using object identifiers or references (pointers). The access by name is location independent (node independent), assuming the name is unique within the application. A hashing function assigns a unique identifier to each name. These identifiers are implemented as indices in a hash table of objects, which is unique for each local database. Since random access is very rapid in main memory, pointers can be followed quickly. Therefore, the hash table contains pointers to the data, rather than the data itself, which eliminates a problem of storing objects of various sizes. Objects can form complicated structures inside the database, referring to each other via references or identifiers. Since references are actually pointers, the use of them is limited to static objects only.

RTDB, following the toolkit approach, allows for separately supplying hashing algorithms including various solutions to handling collisions and allowing for selecting an algorithm offering the optimal distribution of keys in the hash table for a given application.

III. Features

RTDB offers several interesting features, which extend its capabilities and together establish the unique character of that database system.

A. Data Representation

RTDB stores data in the exact form in which it is used by the application, eliminating the need for translating data from one representation to another. Such a translation is needed, for example, if a relational representation is used. All local user processes have direct access to data as opposed to using a server, which introduces unwanted delays. Remote accesses can't be implemented the same way; therefore, a server, acting as a proxy for the remote process, is used.

B. Concurrency Control

A set of operations is provided by the database to allow for database level and object-level locking.

Applications can control concurrency by locking the entire database, which amounts to serial execution of all operations and therefore drastically reduces the costs of concurrency control.

User processes can also acquire single or multiple object locks, examine locking states, and forcibly free locks. Acquiring multiple locks at a time can prevent deadlocks. The capability to unlock objects locked by other processes can prove useful when implementing a mechanism to recover from processing exceptions and processes terminating without releasing all previously acquired locks.

Following the philosophy of a toolkit, RTDB implements basic concurrency control mechanisms but policies must be designed and implemented by the application.

C. Triggers

RTDB offers a mechanism similar to the trigger mechanism typically found in relational database management systems. A trigger can be attached to any object inside the database. It can be defined to be activated upon an object modification or access. Three types of actions can be defined: a) a user-defined trigger handler procedure is invoked, b) a user-specified task is spawned, or c) a user-defined event is broadcast (It is important to note here that broadcast is implemented using a guaranteed delivery protocol.) Handler procedures are executed inside caller process' address space; therefore, they have direct access only to local resources. Typically, they are used to ensure data consistency or for verification. When a required action may take longer and can be conducted asynchronously, a separate task can be spawned to handle the trigger processing. Since events can be delivered to multiple distributed processes, the trigger-induced event allows for having a number of listener processes that can register for the event and perform necessary processing. The triggering process updating an object does not need to know about other processes interested in changes to the object.

D. Attribute- based queries

RTDB incorporates an object retrieval mechanism that is based on object attributes. Each object in the database has a set of binary attributes assigned to it that can be freely manipulated by the application. Therefore, the values of attributes for a particular object can change during the lifetime of the application. The meaning of the attributes is supplied by the application. The database provides operations to select objects based on subsets of attributes. For example, one can use attributes to define classes or states of objects. In such a case, the following queries would be possible:

select class = * **and** state = suspended
select class = dataSource **and** locked != true
select class = pressure **and** alarmRedZone = true

The actual implementation does not use a query language to avoid delays introduced by translating query statements, but rather uses a library of functions enabling definitions of searches by specifying bit masks with included and excluded attribute sets.

This query mechanism is very fast and represents a compromise between a slow and complex, yet powerful query language and solutions with no provisions for system supported queries.

E. Versioning

RTDB has provisions for keeping multiple versions of an object. It allows for having access to a limited history of changes. Such a mechanism can prove useful when examining trends or restoring previous states.

The system also offers a working version of an object, called a shadow. The application can work on updating the shadow of an object and then with one call can replace the object with its shadow. Depending on the application, working on the shadow may require obtaining a lock on the object first to prevent concurrent unsynchronized updates. Having a shadow version frees processes from copying an object to and from the database, and therefore improves performance of the system.

IV. Distributed Database System

Local RTDB memory databases can collaborate to form one distributed database. The set of collaborating databases is defined in a node collaboration list. There are two RTDB mechanisms that help to build distributed systems: transparent access to remote data and distributed synchronization.

Requests to access objects that can't be satisfied locally are broadcast to collaborating nodes. The received response is returned to the user process and a copy is put into a local database (cached). A local cache is implemented to improve performance when accessing static data. The decision to cache an object locally belongs to the application.

Collaborating nodes are also connected via an event bus, which implements a subscribe-publish model of a software bus. Owing to the event bus, events initiated by database triggers are passed between all the collaborating nodes. Processes that have subscribed for a particular event can either wait for notification or check to see that the event has happened, regardless of the originating node of that event.

V. Application of RTDB

The RTDB system has been used in the implementation of the Distributed Monitoring and Control System (DMCS) [6], [7]. The DMCS serves a dual purpose; it is required to monitor and control the environment while performing special tests and measurements of accelerator magnets. Therefore, the system has been constructed as a general-purpose monitoring and control system and a measurement tool at the same time.

The DMCS is a multi-platform system consisting of a set of distributed computers connected via a local area network. Functionally, the system follows the traditional, hierarchical approach: graphical user interfaces and data analysis applications run on workstations under Unix while data acquisition and direct control run on process control computers under the VxWorks real-time operating system. Most of the system components are available on both platforms. System processes, distributed between multiple nodes, communicate via a software bus.

The system configuration is described using the Data Base Definition Language (DBDL). Currently, the system description includes four classes of objects: scans, process variables, devices, and calculations. DBDL enables the user to define all of the objects present in the system. The resulting configuration is loaded to the process control computers for interpretation. All configuration objects are kept in RTDB.

RTDB plays a central role in the system. It resides in each of the process control computers and stores the system configuration, the state of the process under control, test conditions, and measurement results.

The RTDB query mechanism is used primarily to browse through stored objects and select objects of interest to the user via the provided graphical user interfaces.

The trigger mechanism has been used to implement alarms. Upon the modification of an object, a trigger handler procedure examines the object's state and determines if any of the defined alarm values has been exceeded. If it is the case, an alarm event is broadcast.

The trigger mechanism has been also used to implement the software quench detection for the high temperature superconducting leads [9]. Voltages and temperatures along the leads have been defined as quench thresholds. Exceeding one of the defined thresholds is equivalent to detecting a quench, and this results in a request sent to the power control system [8] to ramp down the current. The voltage and temperature values are checked in a trigger handler executed whenever any of the temperatures or voltages is updated in RTDB by a monitoring scan.

The DMCS databases contain a practically constant set of objects and have a well-defined phase of creation, followed only by accesses and modifications, but no insertions. Therefore, instead of chaining, overflow bucket creation or rehashing, the hash table is reorganized, which offers a very good performance for this application.

The dynamic contents of the databases are continuously written to the non-volatile memory by specialized data archival scans.

VI. Conclusion

Memory-resident databases offer an interesting architectural alternative when building multi-process systems. Thanks to short and predictable access times, they seem to be especially suited for real-time applications and data acquisition systems. Moreover, in-memory databases can, similarly to traditional databases, be constructed as distributed systems. As repositories to be primarily accessed programmatically, they can follow an object model rather than a relational model of data.

RTDB, being a toolkit, offers many opportunities for extending and improving. One of the possible development directions is to work on increased robustness of the system. It could be achieved, for example, by monitoring processes to detect process exceptions resulting in blocking the database, so the whole system is not brought to a halt by an errant process. This problem is similar to a classical deadlock caused by a process suspended while inside a critical region.

Memory-resident databases can also be used in multilevel database systems and form, together with object persistent storage systems and relational databases, hierarchical database storage systems [10].

VII. Acknowledgement

Authors would like to thank John Tompkins and Mike Lamm for their support and for making this work possible. Jerzy Nogiec thanks Sergey Sharonov for his friendly remarks during the development of RTDB and Kelley Trombly-Freytag for valuable comments on the text of this article.

VIII. References

[1] http://www.mcobject.com/extremedb.htm
[2] http://www.solidteeh.com
[3] http://www.polyhedra.com
[4] http://www.powerdata.com
[5] http://www.vista-control.com
[6] J. M. Nogiec, E. Desavouret, D. Orris, J. Pachnik, S. Sharonov, J. Sim, J. C. Tompkins, and K. Trombly-Freytag, "A Distributed Monitoring and Control System," International Particle Accelerator Conference PAC'97, Vancouver, 1997
[7] J. M. Nogiec, E. Desavouret, J. Pachnik, S. Sharonov, and J. Sim, "An Open Distributed Monitoring and Control System," Proceedings of the International Conference on Computing in High Energy Physics CHEP'97, Berlin, 1997
[8] J. M. Nogiec, E. Desavouret, "Distributed Power Supply Control and Monitoring System," ICALEPCS 2001, San Jose, 2001
[9] J. M. Nogiec, S. Feher, D. F. Orris, J. Sim, M. Tartaglia, "Architecture of HTS Software Protection System," International Particle Accelerator Conference PAC'99, New York, 1999
[10] J. M. Nogiec, K. Trombly-Freytag, D. Walbridge, "Hierarchical Data Archival System for EMS," PCaPAC 2002, Frascati, Italy, 2002