For all but the hardest of the hard

An off-the-shelf database may be the best route for handling the increasing amount of data associated with real-time systems, argues **Chris Hills**

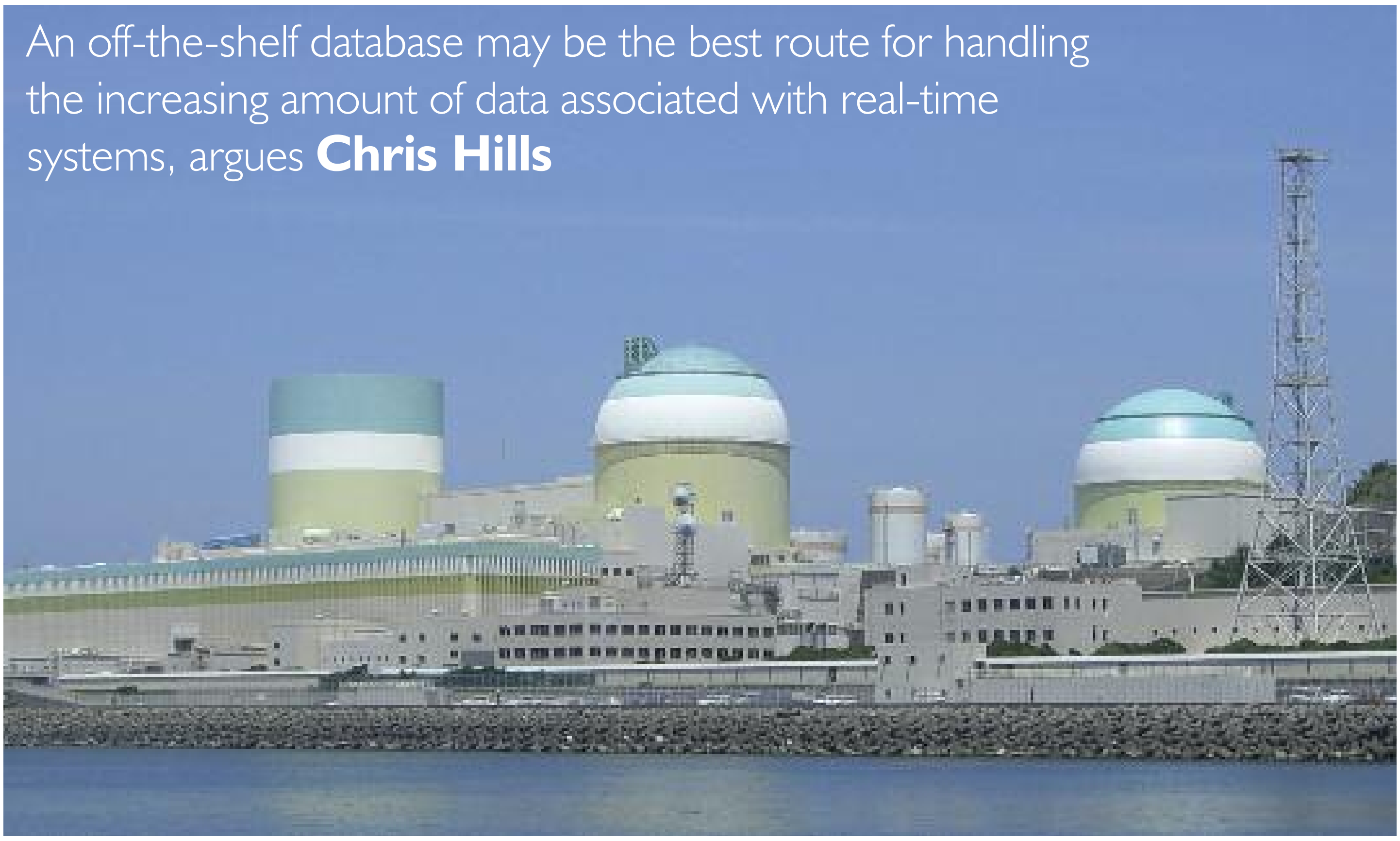
Nuclear power plants cannot tolerate a missed deadline for a critical transaction

Photo: ja:User:Newsliner

Databases in real-time systems sound rather incongruous. Real-time is fast and compact: databases are big, complex, slow and non-deterministic. Yet many of today's real-time systems are handling massive amounts of data. Is it any longer sensible to build system-specific data handling or can an off-the-shelf database management system (DBMS) provide the answer?

What we are looking for is a way of handling input data; correlating, merging or comparing across all data objects and across time, for filtering or analysis. The same data may need to be shared by concurrent tasks that have different functions, time requirements and degrees of importance. This is what a DBMS is designed to do; both traditional databases and real-time databases store, retrieve and manipulate data. The difference is that a real-time DBMS also has to be concerned about time.

Time dependent

For real-time applications, the state of the data in the database has to be as close to the state of the real world as is required by the application (hard real-time is more demanding than soft real-time). Traditional databases are designed for increasing throughput, with performance measured in transactions per time unit.

For real-time applications, however, an important measure is normally the number of transactions that violate the timing constraints: that is transactions that are not completed within a pre-determined deadline. Depending on the environment that the system is working in, the cost of missing this deadline will vary. A third time element is predictability: measures of average response times are adequate in non real-time databases, but for real-time responses have to be predictable to guarantee the completion of time-critical transactions.

To meet these constraints, a real-time database has to avoid using components that introduce unpredictable latencies, such as disk IO operations, message passing or garbage collection. Instead of disks, a real-time database is best implemented as an in-memory system. There is no disk, so no disk IO, and a simpler design than conventional databases reduces message passing.

Since the data in the database have to represent the real world, and the data may be compromised if they are not updated fast enough to reflect real-world events, the DBMS transaction manager has to be aware of time or, at the very least, should provide some way of prioritising the transactions.

Soft and hard

There is a spectrum of real time. At one end is what is generally called hard real-time, the area where a missed deadline for a critical transactions cannot be tolerated, such as in fly-by-wire, drive-by-wire or the control of nuclear power plants. Here, a failure to execute by the deadline could have catastrophic consequences and so designs tend to be custom-made for increased security.

In a hard real-time system, designers often use custom algorithms and schedule excessive amounts of time for a transaction, especially a critical one, to cover a worst-case scenario. These kinds of systems are often

Continued on page 39

Continued from page 37

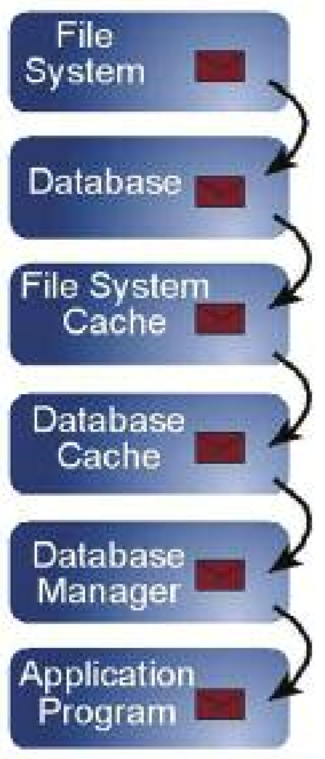

Fig. 1: The journey that a data element makes in a business database

time-driven and time-slice scheduled. For these applications, custom-built data management software and databases are normal.

Off-the-shelf databases, however, are suitable for most real-time systems, generally called soft real-time, where violation of timing constraints results in degraded quality but is to some degree tolerable. Unlike the hard real-time systems, soft real-time systems can be event-driven and priority scheduled. It is in this area that recently there has been a growth in innovation in creating commercially available real-time database systems (RTDBSs) designed to run on off-the-shelf hardware and other elements.

Complex soft real-time systems need databases to support concurrent data access and provide well-defined interfaces between software modules, while supporting levels of performance and predictability lacking in traditional databases. The traditional databases are disk based and cannot achieve predictable response times in the microseconds or milliseconds range.

Main-memory databases can achieve this predictability and main-memory DBMSs are at the heart of real-time databases. They take advantage of the research and development in main-memory database theory and implementation that has been undertaken since the mid-1980s. They also exploit inexpensive memory and the use of 64bit addressing that have become common in embedded systems.

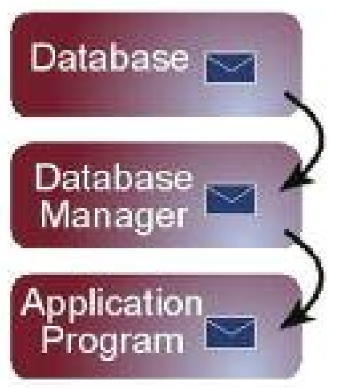

Fig. 2: The journey of a data element within a database designed from the ground up for real-time embedded systems

Commercial databases

A commercial database designed from the ground up for real-time embedded systems can be driven by the need to eliminate performance overhead while providing a predictable and reliable transactional model for applications such as telecoms equipment, factory floor automation systems, process control, zero-latency consumer electronics devices and medical equipment. Unlike traditional systems, it can have a very small code footprint.

Such a system could map its database directly into the application's address space, providing applications with direct pointers to the data elements, eliminating expensive buffer management. Here, a data element takes only a three-stage journey from database to application, in contrast to a traditional database where the journey can include as many as six stages, each with its own variable latency (see Figs. 1 and 2). Access to data is further improved as the associated access structures are placed on the application's stack.

Run-time code can be directly linked with the application, eliminating remote procedure calls, and the execution path can generally require just a few CPU instructions. To improve still further the predictability and performance of database read and write operations, the database need never rely on the operating system's memory management and instead use its own highly optimised memory manager that is responsible for all allocations and de-allocations made by the database. As the database is in main-memory, there would be no bottleneck created by paging data in and out during IO operations.

One of the important facets of a database is it interfacing to the rest of the world. This type of database can have SQL and ODBC interfaces as well as two APIs, one API static and the other a dynamic navigational API that can be configured to fit closely to the application. Optional XML extensions can be used for retrieving XML objects from an external source, creating them in the database, and generating an XML schema for each class in the database.

An operating system is not required as it could run happily on bare bones boards, but if an operating system is available it could take advantage of it.

Fits the task

Database systems are designed to manage persistent data shared among multiple tasks and are built so that transactions maintain the acid (atomicity, consistency, isolation and durability) test of data. Real-time systems add temporal characteristics. In the majority of real-time systems, expired or missed transactions do not lead to catastrophic consequences, they simply have no value. For these soft real-time systems modern commercial main-memory database technology can be effective.

It used to be unusual to buy an RTOS: in-house development was the way to go. Now there is a range of RTOSs available to match specific requirements, allowing developers to concentrate on the differentiating features of their application. With the increasing availability of real-time databases, the same changes are taking place for all except the hardest of hard real-time systems. ■

Chris Hills is founder and CTO of Phaedrus Systems

“It used to be unusual to buy an RTOS: in-house development was the way to go. Now there are RTOSs available to match specific requirements, letting developers concentrate on the differentiating features of their application. With the increasing availability of real-time databases, the same changes are taking place for all except the hardest of hard real-time systems