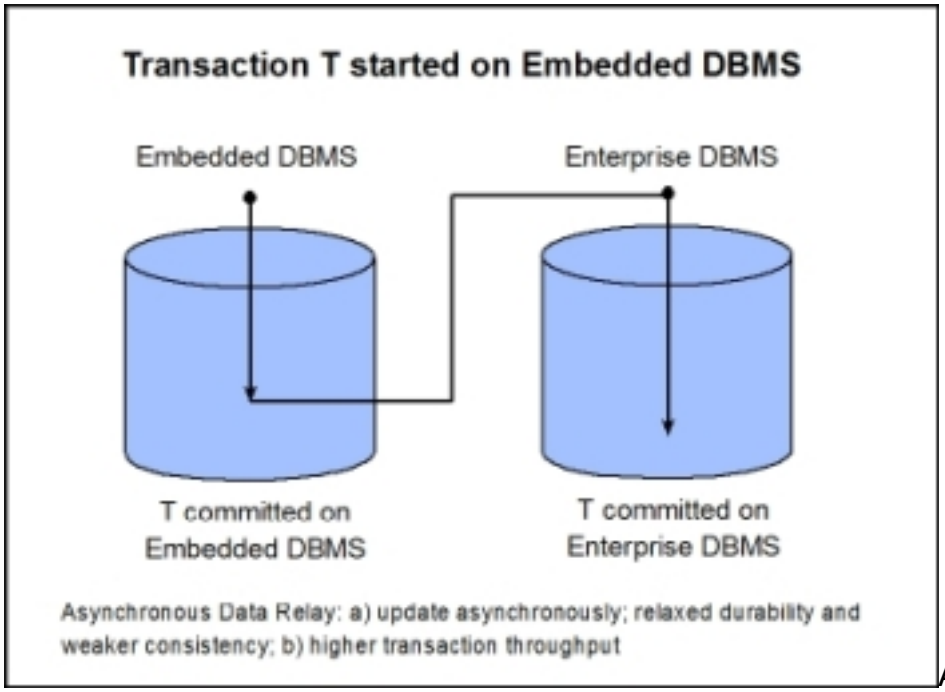
Selective Data Replication Between Embedded and Enterprise Systems

By Steve Graves
McObject



Among wireless systems developers, database management system (DBMS) replication is typically associated with high availability (HA). Deploying real-time DBMS software on a master node, with multiple synchronized copies of that database running on standby nodes, and automatic failover to one of these replicas in the event of the master node's failure, provides for instant recovery and maximum system up-time. This method of ensuring data availability is an essential tool for fault-tolerance in embedded software within switches and other mission critical wireless telecom devices.

Database replication with the goal of high availability is entrenched in the telecommunications field, and is a classic example of a tool you hope you won't need: it's there only in case of disaster.

While replication to provide high availability is widely used, demand has emerged for a new kind of replication. Walls have crumbled between real-time, embedded systems and the outside world of organizations' general IT infrastructures.

Increasingly, the two environments share data, typically with the enterprise system(s) "consuming" data from the real-time embedded application. For example, a wireless telecommunications switch may integrate a real-time database system for purposes including setting up and routing calls, defining services, profiling user

accounts, and storing equipment settings and history. Most of this data is relevant only to the application's core switching task, and therefore "of interest" only to processes running within the switch.

But a sub-set of the data is of interest to external systems. For example, there is typically a "calls completed" data object consisting of information on individual calls such as customer ID, duration, origin and destination points, enhanced services, etc. These details must be consumed by billing, account management and other enterprise applications, in order for the carrier to generate revenue. Typically this requires transferring the relevant data from the real-time, embedded database within the switch to an enterprise database (Oracle, SQL Server, etc.) used in enterprise-wide systems. This is also a form of replication, but with several characteristics that differ markedly from replication that is implemented in order to gain high availability (HA):

Selective. In traditional HA replication, copies of the entire database are kept on replica (standby) nodes. In the "producer-consumer" replication discussed in this article, only a portion of the database is relayed to the consumer. This sub-set can be as fine-grained as certain fields of a single record type. Ideally, the replication mechanism can select only the data that is needed for propagation, in order to minimize network communication overhead and enhance system performance.

Open. Traditional replication is based on using the same database technology on all nodes: an Oracle master database system replicates Oracle DBMS instances on standby nodes, etc. In contrast, this new kind of replication must be open, because enterprise database systems and real-time, embedded database systems are typically quite different technologically, and are provided by vendors that specialize in the respective areas.
How do developers obtain this new type of replication? Usually they've built it, because the capability did not exist in off-the-shelf form. One "homegrown" strategy uses the object notifications (aka triggers) feature of real-time database systems to inform the application when an object changes. Application logic can then determine whether the change meets criteria for forwarding to the enterprise DBMS. A major drawback to this approach is that it is transaction-ignorant. To protect data integrity, DBMSs typically group updates into units (called transactions) in which all changes succeed, or fail, together. In a system using object notifications to implement replication, a transaction involving 15 updates might invoke 15 event handler processes, which the application must re-assemble into a transaction. This entails complex code, and burdensome (in terms of CPU cycles) additional processing.

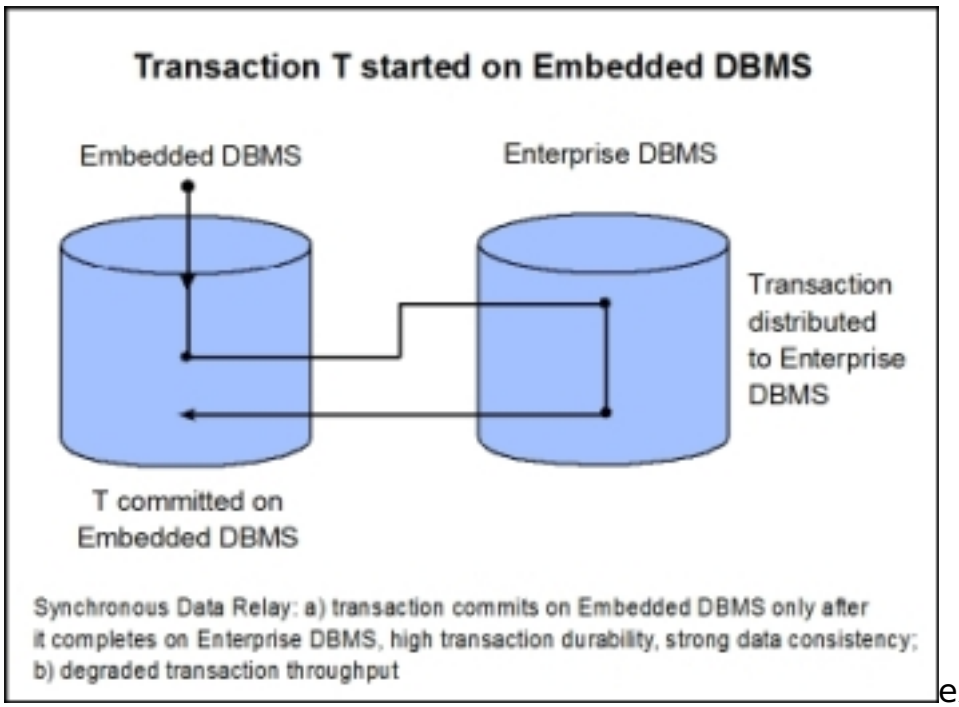
Another strategy is keep a record of database changes and their transactional context in one or more database classes (tables) that are set up expressly for that purpose. At some specified interval, an application process polls/queries the table(s) in chronological order, using an index, to find new updates. This "squirreling away" of information actually closely resembles what occurs in a DBMS transaction buffer. But it necessitates populating the special-purpose table(s) while the DBMS is doing its normal work of database transactions, effectively doubling the work.

As noted above, both of these approaches duplicate work that is already being done by the real-time database system. This wastes processing resources (CPU cycles and RAM) and also begs the question: if the building blocks of selective embedded/enterprise replication already exist in the real-time database system, why should a developer have to re-create them?

When enough developers find they are re-inventing the wheel and coding software that meets a common need, a vendor typically emerges to provide an off-the-shelf solution. In the case of embedded-to-enterprise replication, McObject has addressed this need with its eXtremeDB Data Relay technology. eXtremeDB is McObject's real-time embedded database system, created explicitly for embedded software applications and used widely in telecom/networking equipment. At its core, eXtremeDB offers a streamlined in-memory database system (IMDS) architecture that eliminates disk I/O, caching logic and other overhead, in order to provide the fastest possible performance, with a small processing footprint. For most of its history, eXtremeDB has been available in a High Availability edition, providing the "traditional" replication described earlier in this article.

Data Relay is a more recent addition, and is included as a feature of the eXtremeDB Transaction Logging Edition. Data Relay opens up eXtremeDB's transaction buffer to provide highly selective and efficient embedded/enterprise replication. For every object affected by a transaction, the buffer already provides a coded value that indicates whether the operation was: an insert, update, or delete. With Data Relay, applications use a familiar database "cursor" to iterate over objects in the buffer and extract the coded value. Based on this reading, the application – via the cursor – can take the next step of drilling down into the object within the buffer, analyzing changes and determining whether to propagate the change to an enterprise DBMS or other enterprise software.

Data Relay is designed for fine-grained selection of data for replication, with maximum efficiency. For example, in the case of updates (changes to existing records), Data Relay utilizes a bitmap within the buffer that indicates the fields that were affected. If an object has 200 fields, the bitmap enables the application to read, and relay, only the pertinent ones. This efficiency is particularly important in maintaining the real-time system's performance when data-sharing is synchronous – that is, when the real-time database transaction is committed only after relevant changes are selected, propagated and saved on the external DBMS.



eXtremeDB Data Relay also offers an asynchronous mode, in which transactions can commit on the real-time system's eXtremeDB database before changes are saved on the external system. Synchronous data-sharing guarantees consistency between eXtremeDB and the external database, but it ties the eXtremeDB in-memory database to the (slower) performance of the external database. In contrast, the more “relaxed” asynchronous approach largely sustains eXtremeDB’s natural performance. Synchronous Data Relay mode is useful when eXtremeDB is employed as a predominantly read-only high speed cache to an RDBMS, with occasional updates. Asynchronous Data Relay mode may be favored when high throughput within the embedded real-time system outweighs the need for synchronization between the embedded and enterprise data stores.

Data Relay uses eXtremeDB's existing transaction buffer to support synchronous relaying, but provides a separate, larger buffer to hold asynchronously processed transactions. This bigger pipe is needed because with asynchronous relaying, data needs to stay in the buffer long (potentially much longer) after eXtremeDB is done with it, because eXtremeDB is going to be executing transactions much faster than the enterprise DBMS. Also for this reason, Data Relay’s buffer for asynchronous transactions can be backed up by an overflow file (which is never necessary for synchronous relay).

Due to likely heterogeneous replication scenarios, eXtremeDB Data Relay leaves it to the developer to add data transport and any needed conversion to enterprise DBMS formats. Data Relay provides fine-grained, real-time data selection, and access to this data in binary form. These pieces of embedded/enterprise replication are likely the most time-consuming to develop, and potentially expensive in performance terms. Due to the specialized nature of this new type of replication, traditional HA replication schemes do not provide a practical or cost-effective solution. As more real-time systems in telecom/networking need to monitor the real-time database within their embedded software, and transfer selected changes to the outside world of enterprise applications, developers will increasingly seek a

solution like Data Relay that helps to preserve performance and reduce cost.

Steve Graves is co-founder and CEO of McObject. Learn more at www.mcobject.com [1].

Links:
[1] http://www.mcobject.com