the Availability Digest

www.availabilitydigest.com

Raima's High-Availability Embedded Database

December 2011

Embedded processing systems are everywhere. You probably cannot go a day without interacting with dozens of these powerful systems. Software applications embedded in microprocessor chips control airplanes and toasters. They provide the intelligence for our smart phones and GPS devices. Manufacturing robots and patent monitoring systems depend upon them. The average car today is run by about forty microprocessors with forty million lines of code.

Many of these embedded applications depend upon sophisticated databases. RDM Embedded (RDMe) from Raima, Inc. (www.raima.com) fulfills this need with a highly available embedded database.

RDMe Database Fundamentals

An effective embedded database must meet several criteria:

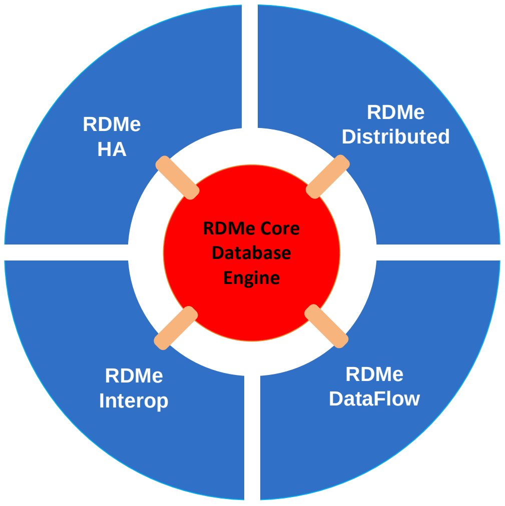
- *It must have a small footprint*. RDMe can use as little as 400K of RAM memory.
- *It must be fast.* RDMe does not depend upon disk accesses.
- *It must be reliable*. RDMe provides transaction semantics and mirroring to protect data.
- *It must be scalable*. RDMe's Database Union and Multi-Version Concurrency Control (MVCC) features let applications work seamlessly across partitioned databases.
- *It must be predictable.* RDMe's core database uses the network model with fixed length records that make access times and database memory sizes easily predictable.
- *It must provide a low level of control*. The RDMe API includes more than 150 low-level control functions implemented in C.
- *It must optionally provide high-level access.* RDMe databases are accessible via C++ object models, SQL statements, and ODBC, among others.
- *It should allow geographical distribution of data*. RDMe's replication facility and remote login capabilities let RDMe be the database in highly distributed peer or hierarchical applications.

RDMe Architecture

The basic RDMe database functionality is implemented in its core database engine. This engine is suitable for many applications and has been put to use in hundreds of thousands of installations over the last twenty-five years. It has served many industries including aerospace, automotive, business, finance, government, healthcare, manufacturing, and telecommunications.

Raima offers several extensions to its core database engine. These include:

- **RDMe High Availability** – RDMe's goal is to achieve five 9s of reliability.
- **RDMe DataFlow** – An RDMe core database can be replicated to one or more external SQL databases.
- **RDMe Distributed** – An RDMe database can be partitioned across multiple locations and viewed as a single database.
- **RDMe Interop** – External applications can access an RDMe database via ODBC, JDBC, and ADO.NET, and via a browser over HTTP.



RDMe Extensions

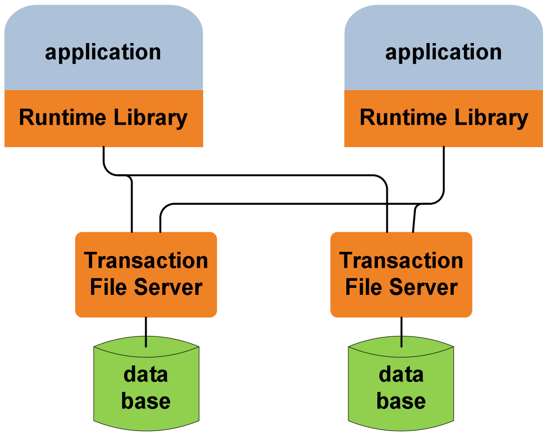
The RDMe Core Database Engine

The RDMe core database engine comprises the database itself, a Transaction File Server to manage the database, and a Runtime Library that applications use to issue database commands.

Database Models

Network Model Database

To achieve maximum efficiency in the use of space and time, the underlying database architecture of RDMe is a network model database. It is based on *owner/member* relationships. A file comprises an owner record and many member records. The members form a linked list, with each member record pointing to the previous and next member record as well as to the owner. The owner record points to the first and last member record in the list.



RDMe Core Architecture

All records in a file are fixed length. They are organized into fixed-length pages with multiple records per page. It is the page that is the block that is written to and read from disk.

The core database supports five types of files:

- *Database Dictionary Files* which contain the data definition schema for each file.
- *Data Files* which contain the file data.
- *Key Files* which contain B-tree indices on fields within a file.
- *Hash Files* which can provide quicker lookup of records than going through a B-tree index providing key ordering is not necessary in the application.
- *Vardata File* for holding variable length data that is referenced by a pointer in a data record.

Files may be fully resident in memory. A file may also be resident on disk, with memory being used as cache to hold the most recently used records from the disk-resident file. To maintain persistence, an application can "sync" the memory-resident content of a file at any time by writing it to disk.

Thus, a core RDMe file is a linked list of fixed length records. The network model is extremely efficient when records must be read in sequence.

Relational Model Database

Through the use of its Key Files, the RDMe database is extended to a relational database that sits on top of the core network model database. Records may be selected in key order via the keys in the Key Files. This allows the database to be used as a SQL database. SQL DDL statements are translated into core DDL statements and stored in the core Database Dictionary Files.[1] Relational equi-joins, based on primary/foreign key relationships, are actually implemented in the core database as network model sets, resulting in very fast join processing in SQL.

Runtime Library

The RDMe Runtime Library provides applications with the functionality needed to make effective use of the RDMe database. A copy of the Runtime Library is bound into each application that needs to access RDMe. Runtime Library functions include:

- *Database Control* – Add and delete tables, open and close files.
- *Record Set Control* – Create and delete records, connect or disconnect records from sets.
- *Data Manipulation* – Read or write individual fields or entire records.
- *Locking* – Lock records for shared reading or exclusive writing.
- *Transaction Control* – Begin, commit, and abort transactions.
- *Navigation* – Key lookup, hash lookup, and scanning.

Transaction File Server (TFS)

The Transaction File Server (TFS) serves the role of a database controller, much like a disk controller manages disk activity. An RDMe configuration can have multiple TFSs, and they all operate independently. Though a TFS can manage multiple disks, typically only one disk is assigned to each TFS to achieve the maximum parallelism for disk activity.

Any number of applications may connect to a specific TFS through their Runtime Libraries. It is the responsibility of the TFS to execute commands invoked by applications via their calls to the Runtime Library functions.

A file partition is always contained on one disk. Therefore, all commands executed against a given file (or partition thereof) always flow through the same TFS. The TFS manages the cache and the locks for its files.

Distributed Processing

In addition to multiple TFSs on the same computer, there can be TFSs on multiple computers. Any application can connect to TFSs on any computer. Therefore, the database may be distributed over several computers that may, in fact, be geographically distributed.

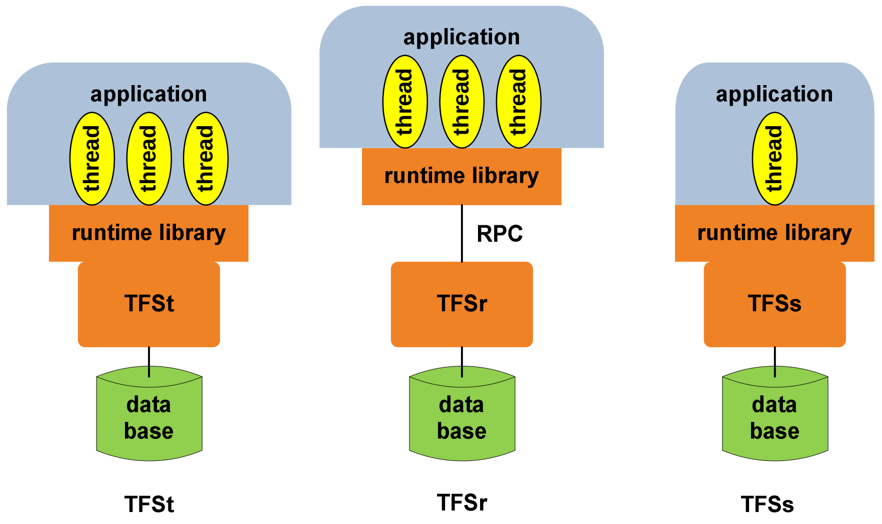
Applications connect to the TFSs via TCP/IP. A TFS is identified via a domain name and a port. If an application is connecting to a TFS on its own computer, the TCP/IP stack is optimized for local communication.

[1] For a benchmark comparison of network versus relational databases, see the Raima white paper entitled Database Management in Real-time and Embedded Systems.
http://www.raima.com/wp-content/uploads/Database_Management_in_Real_Time_and_Embedded_Systems.pdf

TFS Versions

There are three different versions of TFS:

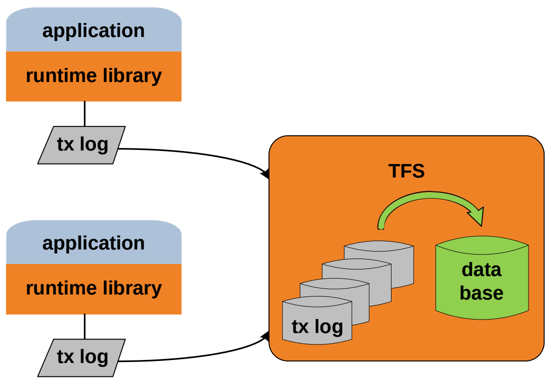
- *TFSt* is a full-featured TFS that links directly to an application. It is multithreaded and can execute several runtime commands simultaneously. It is optimized for a multicore environment.
- *TFSr* has the same functionality as TFSt but applications connect to it via RPCs (remote procedure calls). This capability is used if the application and the TFS are on different computers.
- *TFSs* is a single-threaded TFS that is intended for batch processing. It is not thread safe, and the database should be backed up before processing it with TFSs.



TFS Versions

Transaction Control

When an application issues a *begin transaction* directive, the Runtime Library initiates a transaction. As data manipulation commands are issued, TFS is notified and locks the records that are being changed. The Runtime Library keeps a copy of the changed records.



RDMe Transaction Processing

Upon the receipt of a *commit transaction*, the Runtime Library bundles all of the pages that have been newly created or updated by the transaction and sends these as a transaction log to the TFS. The TFS updates its cache with the changes and releases the locks on the changed records. It writes the transaction log to a disk-resident log for persistence. At this time, the transaction is safe-stored and is considered committed. The commit directive issued by the application is completed with a status response.

Periodically, TFS will update the disk-resident database with the transaction logs that have accumulated since the last synchronization point. The synchronization interval is configurable from ten milliseconds to ten seconds.

If a transaction affects files managed by multiple TFSs, the changes are sent as separate transaction logs to each involved TFS. However, there is no two-phase commit protocol between the TFSs.

SQL Access

A fully functional SQL implementation is much too massive for an embedded system. Fortunately, many of the SQL functions are not typically needed in an embedded application. Raima has implemented a subset of SQL that provides the necessary functions required by most embedded applications.[2]

Raima's abridged SQL does not include Grant/Revoke security statements, views, triggers, or dynamic DDL. It typically executes precompiled SQL statements but, as an option, Raima provides a SQL compiler.

The SQL engine is linked with the RDMe Runtime Library and uses the underlying RDMe core database to execute SQL statements. A SQL application can interact with multiple TFSs on the same or different computers.

Operating System Support

RDMe runs under many operating systems, including Linux (many flavors), Windows, HP-UX, Solaris, iOS, AIX, and Mac OS.

RDMe Extensions

As described earlier, Raima offers several extensions to RDMe.

RDMe High Availability

RDMe provides many features to achieve high availability of its database. Raima's goal is to achieve five 9s of availability (downtime of about five minutes per year) and 100% protection of the data in the database.

Transaction Management

RDMe availability starts with its transaction management capabilities. Once a transaction is committed, it is durable. Thus, the database remains intact following a system failure. Upon recovery, the safe-stored transaction logs are read to replay any transactions that had not yet been materialized at the time of the failure.

Hot Backup

RDMe provides a hot backup facility to take full or incremental snapshots of active files and to write them to an offline medium. The backup facility does not interfere with normal processing, which continues unabated during the backup. Database backups allow the database to be restored should the disks suffer damage.

Mirroring

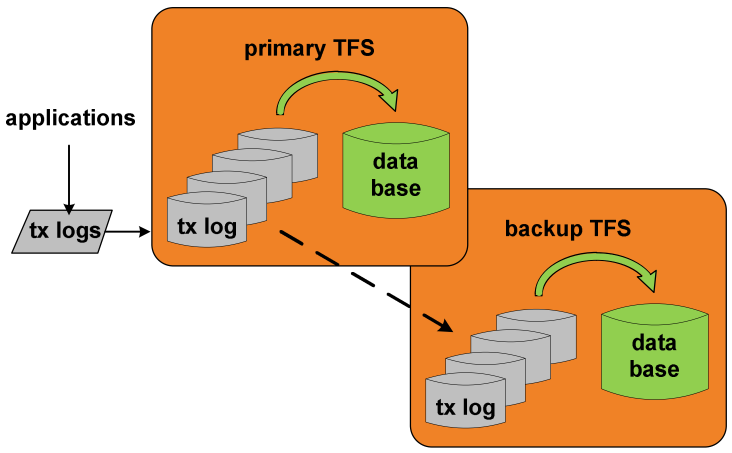
To provide rapid recovery from a total system failure, the RDMe database can be mirrored. Mirroring provides a byte-by-byte copy of the database on a backup system. The backup system can then be put into service rapidly should the primary system go down.

Database mirroring is accomplished by copying the transaction logs from the primary system to the backup system. The backup system replays the transaction logs against its copy of the database to keep the two databases synchronized.

[2] See Is Using SQL in an Embedded Computer Application Like Trying to Squeeze an Elephant into a Mini?, *Raima White Paper*.
http://www.raima.com/wp-content/uploads/sql_in_an_embedded_application.pdf

Mirroring can be either synchronous or asynchronous. With synchronous mirroring, both systems will synchronize their databases with the accumulated transaction logs simultaneously. Database synchronization is not considered complete until both systems have materialized all of the data changes in the transaction logs. In this case, the primary and backup databases are always exact copies of each other.

With asynchronous mirroring, the backup system will synchronize its database independently of the primary system. The backup database will be out of synchronization with the primary database by some small increment of time.



RDMe Mirroring

Raima supports third-party HA Managers. HA Managers run on each system and coordinate the mirroring process. It is the responsibility of the backup HA Manager to determine when the backup system should take over. A Raima-supplied HA Manager is scheduled to be available in the future, along with user-defined callouts for taking specified actions on certain failures.

RDMe DataFlow

Via RDMe DataFlow, the contents of an RDMe database can be distributed to other systems for their use. RDMe DataFlow uses data replication rather than mirroring to do this. As opposed to mirroring, RDMe DataFlow replicates core database actions such as create, update, and delete rather than byte changes. The Runtime Library creates replication logs containing these actions for each transaction when configured to do so.

The replication logs are sent asynchronously from the primary system to one or more target systems. Resident on each target system is a Replication Client that is responsible for mapping the replication log to the target system's database and for replaying the replication logs to that database.

A primary use of RDMe DataFlow is to move data from remote non-indexed databases to a heavily indexed persistent query database. For instance, it can feed SQL databases such as MySQL, SQL Server, Oracle, or another RDMe SQL database. It can also be used to aggregate data from several sources onto a single target. For instance, data from multiple remote sensors can be aggregated in the database of a central control computer.

RDMe Distributed

RDMe Distributed provides a unified view of a geographically distributed partitioned database. The partitions are identically structured, and each partition can reside on a different computer. Through RDMe's Database Union feature, an application can access the distributed partitions as a single database without being aware of the fact that they reside on different computers.

RDMe uses Multi-Version Concurrency Control (MVCC) to present a consistent snapshot of the database to a reader even while the database is being actively updated. This allows readers to view the database without having to lock records in order to maintain a consistent view. As a result, MVCC read activity is transparent to applications that are actively updating the database.

RDMe Interop

RDMe Interop allows other systems to interoperate with an RDMe database via standard protocols. RDMe Interop supports ODBC, JDBC, and ADO.NET. RDMe databases can be read and updated with Internet browsers over HTTP using RDMe's MicroHTTP Server.

Use Cases

A series of detailed use cases for RDMe can be found in the Raima white paper entitled RDM Embedded 10.1 Architecture and Features.[3]

Summary

Embedded systems are all around us and control many aspects of our lives without our ever thinking of them. Some of these are simple systems that control our coffee makers and dish washers. Others are quite complex, controlling manufacturing processes and jet airplanes.

Complex embedded applications often require sophisticated, efficient, and high-performing databases. Raima's RDMe database fulfills this need. It can be configured as an elemental network database requiring less than 400K bytes of memory all the way up to a sophisticated SQL relational database.

RDMe has many features to support high-availability applications with a goal of achieving five 9s of availability. It can be geographically distributed in peer and hierarchical architectures.

RDMe is a mature database. It has been used in hundreds of thousands of embedded installations over the last twenty-five years.

[3] http://www.raima.com/wp-content/uploads/RDMe-10-1-Technical-Summary.pdf