Building an Application-Specific Database Server Using RDM Server

A Raima Inc. Technical Whitepaper

Published: August, 2008
Author: Randall L. Merilatt
                CTO
Copyright: Raima Inc., All rights reserved

Abstract

How do I turn my application into a database server? This paper discusses and describes how to turn your application into a database server using RDM Server. Main reason to do something like this is to remove the process boundaries between the data manager and your application to gain performance. Have a look.

This article is relative to the following versions of RDM:

✓   RDM Server: 6.0, 6.1, 7.0, 8.0, and 8.1

Contents

Introduction

RDM Server provides a powerful capability that allows you to create your own application-specific, multithreaded database server. Figure 1 below illustrates the architecture of a typical RDM Server client-server application. The shaded areas represent RDM Server components.
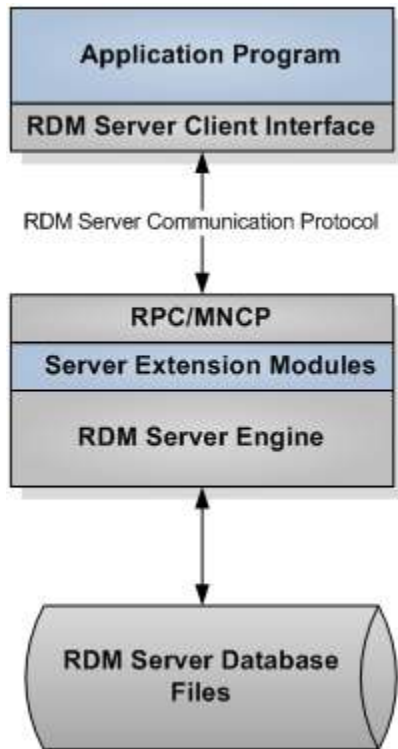

Figure 1, RDM Server Client-Server Architecture

The sample application consists of the following:

- ✓ The client-side program that accesses the server through the RDM Server client interface libraries (for example, SQL and/or remote procedure calls)
- ✓ The application-specific server extensions that run on the server

Communication is provided by the RDM Server Multiple Network Control Processor (MNCP).

In the classic architecture, communication with the RDM Server is restricted to a separate RDM Server process. This process uses the Remote Procedure Call (RPC)/MNCP interface, and RDM Server continues to support this architecture. However, a powerful alternative is now available providing more flexibility in the use of RDM Server as an embedded database system. In this extended architecture, you can link your application directly to the RDM Server engine, which allows your application to become the database server. Refer to Figure 2.
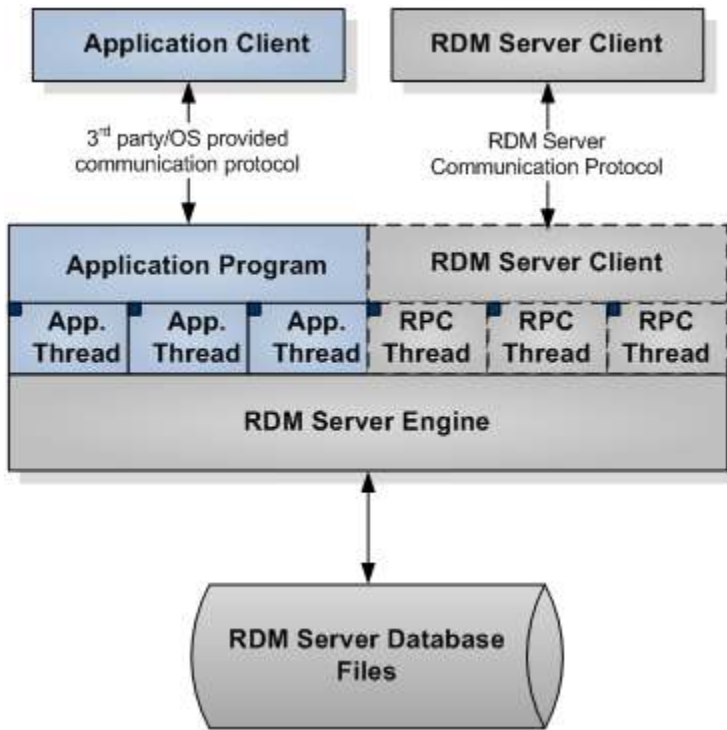
Raima

Figure 2, Application Server Architecture

As an application-specific database server, you choose the type of remote communication protocol you will support. For example, ;you might use a protocol provided by the operating system, or use a third-party supplier such as COM/DCOM on Windows NT or JAVA. The communication would occur under the control of your application, completely outside of RDM Server. To fully benefit from the multithread RDM Server environment, your application would launch and manage its own threads, each controlling one or more RDM Server login sessions.

When your application launches RDM Server (through a call to the RDM Server **s_startup** function) you can choose to have RDM Server start its RPC/MNCP threads (shown as the dashed boxes in Figure 2). This gives any standard RDM Server client program (for example, third-party ODBC tools) access to the RDM Server database.

Standalone (non client-server) applications that require the power and performance of a full multithreaded database engine will benefit from not having to incur the performance and memory penalties associated with access to a separate process (for example, the RDM Server program) through the RDM Server shared-memory NCP.

An interesting example of this involves database support for Internet web server applications as shown in Figure 3 (shaded boxes are representative resources used by the application extensions). With most Internet web server software development kits (such as Microsoft's Internet Information Server or Netscape Server), you can develop server extensions much like those in RDM Server. To access a typical DBMS database server, a web server extension must connect to that database server and access the database information using the network protocols supported by that DBMS. With RDM Server, the database engine can be directly linked with the Internet web server thus providing more efficient, faster access to the database information. Very powerful

database-enabled web applications can be built using the RDM Server embeddable multithreaded database engine technology.
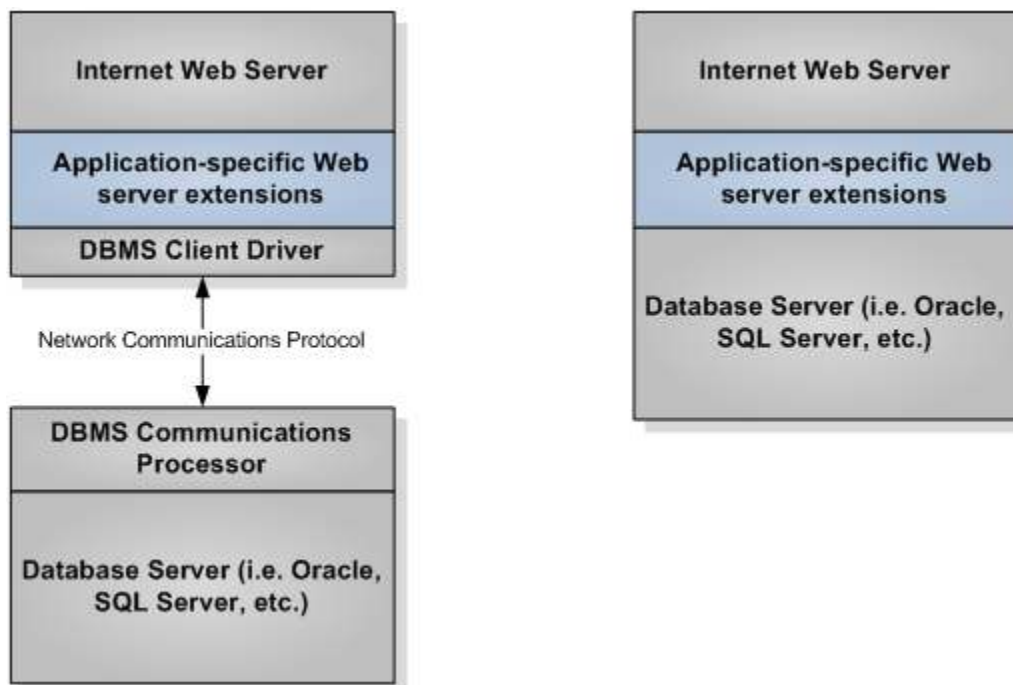

Figure 3, Web Server Architecture

Application Database Server Development Guidelines

Two administration functions control the startup and termination of the RDM Server database engine, **s_startup**, and **s_terminate**.

After invoking RDM Server by calling **s_startup**, you can initiate any number of login sessions using the **s_login** function. (Note that the server name parameter to **s_login** is unused when the function is called from an application server.) Use of multiple threads will result in better scalability and system throughput. Each thread will issue its own **s_login** calls. Note that it is important that the calls to RDM Server functions are serialized for a g iven session. You can accomplish this by issuing all calls for a given session from the same thread. Although you are free to directly call the thread management functions provided by your operating system, you can also use the RDM Server Resource Manager functions and benefit from the platform independence they provide.

After all sessions have logged out (**s_logout**), call **s_terminate** to shutdown the RDM Server database engine. It is truly that simple. Besides the calls to **s_startup** and **s_terminate**, everything else works just the same as with the client-server architecture, except that your entire application performs like an extension module without having to be accessed through the RDM Server RPC interface. Of course, as an application server you are also responsible for managing the communication with any client computers.

If you need standard RDM Server client applications and third-party ODBC access to the RDM Server engine running under your application server, two additional functions must be used. To initiate the RDM Server RPC subsystem, call function **s_startRPCThreads** (after **s_startup**). You must pass to this function the name of the server, the number of session threads to be used to handle client login sessions, and the address of a short

variable used to indicate that a system shutdown has been requested. To shutdown the RPC subsystem, call function **s_endRPCThreads** . This function must be called before calling **s_terminate**.

These functions are summarized in the following table.

Table 1 - RDM Server Control Functions

Function	Description
s_startup	Startup the RDM Server database engine.
s_startRPCThreads	Startup the RDM Server RPC (and NCP) subsystem and launch session threads.
s_endRPCThreads	Shutdown the RDM Server RPC subsystem.
s_terminate	Shutdown the RDM Server database engine.

Use of these functions is illustrated in the simple example code shown below. This example starts up a basic RDM Server, server.

```
#include "velocis.h"
static RDSLOGFCN MessageConsole;
static short shutdown_flag;
/* Basic RDM Server database server
*/
void main(
    int argc,
    char *argv[])
{
    short stat;
    char *catpath = NULL;
    char *server = "RDM Server";
    if ( argc > 1 ) {
        /* server name is first argument, default = "RDM Server" */
        server = argv[1];
        if ( argc > 2 ) {
            /* alternate catalog path is second argument */
            catpath = argv[2];
        }
    }
    stat = s_startup(catpath, MessageConsole, LOG_ALL);
    if ( stat != S_OKAY ) {
        printf("Unable to start RDM Server engine, status = %d\n", stat);
        exit(1);
    }
    rm_interrupt(&shutdown_flag);
    stat = s_startRPCThreads(server, noSessionThreads, &shutdown_flag);
    if ( stat != S_OKAY ) {
        printf("Unable to start RDM Server RPC server, status = %d\n", stat);
        exit(1);
    }
    while ( ! shutdown_flag )
    rm_sleep(5000L);
    s_endRPCThreads();
    s_terminate();
}
```

The program takes two command line arguments. The first is the name of the server. The second argument, if specified, is the directory path to the RDM Server catalog directory.

The call to the **s_startup** function has three parameters:

1. The first is a string containing the catalog directory path. If NULL, RDM Server will get the path from the CATPATH environment variable or, barring that, the current directory. An error is returned if the catalog cannot be found.
2. The second parameter is the address of your own RDM Server message log function. This function will be called to process every error or information message generated by the RDM Server database engine. If you specify NULL, the default behavior is to output each message to stdout and to the rds.log file.
3. The third parameter specifies the message classes to be logged. This example specifies that all messages will be logged (refer to the s_startup function description for a list of all message types). Note that if s_startup returns an error, the messages relating to that error will have already been logged.

The log function itself is shown below, and it is passed three parameters.

1. The first indicates whether the function will open its log message processing, just log a message, or close the message log. The options are indicated by the constants RDSLOG_OPEN, RDSLOG_MESSAGE, and RDSLOG_CLOSE.
2. The second parameter specifies the message type, such as error, warning, or information message.
3. The third parameter is the message string. This particular example is trivial but illustrates the basic construction.

```
/* Log RDM Server console message
*/
void MessageConsole(
    RDSLOG_CTRL fcn,          /* call type: open, close, message */
    short type,               /* message type */
    char *msg)                /* message to be logged */
{
    static FILE *errlog;
    switch ( fcn ) {
        case RDSLOG_OPEN:
            errlog = fopen("RDM Server.log", "w");
            break;
        case RDSLOG_MESSAGE:
            if ( type == LOG_ERROR ) {
                /* make sure this gets noticed */
                fprintf(errlog, "******* ERROR! ******* ");
                printf("******* ERROR! ******* ");
            }
            fprintf(errlog, "%s\n", msg);
            printf("%s\n", msg);
            break;
        case RDSLOG_CLOSE:
            fclose(errlog);
            break;
    }
}
```

After the RDM Server engine has been successfully started, function **rm_interrupt** is called to enable the capture of any user-initiated program interrupts. The single parameter specifies the address of a short variable (**shutdown_flag**) that will be set to 1 when a user-interrupt occurs.

The call to function **s_startRPCThreads** invokes the RDM Server RPC/NCP subsystem specifying the server name and the number of session threads to be launched. The session threads form a pool of threads that will handle the execution of one or more user login sessions. The system is designed so that all active login sessions can be

handled by a fixed number of session threads that is smaller than the actual number of active login sessions. If the number of threads value is zero then the default number from the **rdm server.ini** file will be used. The third parameter is a pointer to a short variable (again we're using **shutdown_flag**) that will be set to 1 in the event that some administrator user has requested a system shutdown.

The system now begins an indefinite series of 5 second naps ended only when the **shutdown_flag** has been set either through a user-interrupt or by some remote administrator requesting a system shutdown. Once the **shutdown_flag** is set, the program calls **s_endRPCThreads** to shut down the RPC/NCP subsystem and then calls **s_terminate** to shut down the RDM Server database engine.

Conclusion

And that concludes our discussion of how to build an application-specific database server with RDM Server. Besides just having it sleep, the program might contain a user interface that could, for example, perform many of the administration duties handled by the **admin** or **dram** utilities. There may also be application-specific administration capabilities that you would like to restrict to the server computer itself. Any single computer application that requires a high performance database engine is an ideal use for this technology. Examples include tape backup systems, real-time event monitors, and telephone switching systems.

The ability to directly link a full-featured, high-performance database management engine to an application is one of the unique features of Raima's RDM Server product making it the premier commercial embedded DBMS offering available today.

Contact Information

Website: http://www.raima.com

WORLDWIDE

Raima Inc.
720 Third Avenue, Suite 1100
Seattle, WA 98104
Telephone: +1 206 748 5300
Fax: +1 206 748 5200
E-mail: sales@raima.com

EUROPE

Raima Inc.
Stubbings House, Henley Road
Maidenhead SL6 6QL, United Kingdom
Telephone: +44 1628 826 800
Fax: +44 1628 825 343
E-mail: sales@raima.com