Paradox: A Cheap Alternative to Client/Server?

William Rouck

Client/Server Database Management Systems (DBMSs) are very expensive. But when you need to provide database access to a number of different people scattered across the company or across the globe,you really don't have any choice. Or do you? In this article, you'll discover a very inexpensive alternative to Client/Server DBMSs that, under the right circumstances, can give you exactly what you need.

WHEN considering a database platform to service a Wide-Area Network (WAN) user base, a systems development shop will typically look at both what could be done and what makes sense economically. The developers begin the process by considering the best possible way to fulfill the business requirements. The solution often rings up a high price tag in terms of time and resources. Management will always step in and take into account the concept of time and dollars. When these resources are short, new options must be generated, and those options do exist. Delphi supplies its low-end packages with the ability to deploy a sound WAN-deployed application, but only when developers give special consideration to issues such as network traffic and application design.

A client/server DBMS is usually the best WAN solution. Whether you use Interbase, Oracle, Sybase, or Microsoft SQL Server, a well-designed client/server system provides much more power and stability than a desktop database package.

But, as any client/server veteran will tell you, this power comes at a price. Considerable resources must be dedicated to the tasks of database administration and testing. Of course, if the system happens to be your multinational company's 24/7 accounting system, you gladly spend the money and resources on a solid client/server system as a matter of survival. However, if the system is for the guy down the hall to automate his index-card inventory of small purchase orders, you'd almost certainly prefer a desktop database. If the system's scope falls somewhere in between, you have more options.

In this article, I'll present a solution using Paradox for a low-load but nationwide database application. You might also find this material applicable in cases where the shared data need is local, but remote access would help for a few remote users. It can be done cheaply and, if designed correctly, can serve as a springboard for a true client/server system when the need arises.

The politics of going national

Planning a nationwide system isn't easy, because your system must work with the lowest common denominator. In a large and widespread organization (departments of the Federal government, for example), the level of modernization will vary by location. Some offices are well-equipped with the latest Pentium workstations running Windows NT, while others are skating by with 486s running Windows 3.11 in a peer-to-peer configuration. The bottom line for you is that the application has to work with all of them. This will mean writing a 16-bit version of the system without the power of today's latest tools, but it can be done.

Part one: Application and data design issues

The first step in developing a Paradox-based Delphi application for deployment over a WAN is to research your network. Become familiar with how your nationwide network is configured and its limitations. Most importantly, identify those responsible for the various facets of its maintenance. Create a contact list of the home-base administration Workgroup, as well as the remote administrators. Bounce your ideas off of them to see whether they can find any obvious problems with your connection needs. Keep them informed of your purpose and progress throughout your development cycle in order to gain cooperation when you need it during testing and deployment. You'll need it when dealing with Windows Internet Naming Service (WINS) server issues and network protocol questions.

The second step is to know how your database will be stored. I recommend that you host your database on an NT server. When it becomes time to upscale the

application, you can be sure that a suitable database back end can be found to run on your hardware and software platform. Also consider how you'll access your server in a crisis. Some organizations don't allow physical access to file servers freely, so make sure you can get to yours when you need to.

The third step is the required-reading phase. In order to create a stable multi-user Paradox application, you must be fully aware of the particulars involved. Your first source is Borland's Technical Information (TI) Document #2989, obtainable via search on their Web site at http://www.borland.com. This document, titled "BDE Setup for Peer-To-Peer (Non-Dedicated) Networks," will outline the requirements for simultaneous connections to your Paradox database. It also alerts you to requirements that depend on your mix of 16-bit and 32-bit BDE systems. For example, 16-bit BDE systems don't support aliasing by the Universal Naming Convention (UNC), so you'll need to be able to map a drive letter from the remote site to your database directory. Check out T12770, titled "BDE Frequently Asked Questions." Also required is a thorough study of the Borland newsgroup forums as a get-to-know-you session with Paradox problems and common errors.

The fourth step is your system design phase, during which you can't forget that you're shoe-horning a desktop database platform into a client/server replacement. Every design decision you make must include an answer to the following two questions:

- What is the network traffic implication of this decision?
- How will this decision affect performance for the low-powered users?

Data issues

While designing your application, you need to understand how Delphi is going to read your Paradox data a thousand miles away through a series of hops and lost packets. The client will consider the remote directory to be a drive from which the tables will be read. Your Delphi program will open a form with a filtered table, displaying a grid of stock items for the user to choose from.

In a client/server system, the client sends SQL statements over the network, and the server does the filtering and returns a result set. However, the Paradox system will read the remote tables as it would a local set of tables-reading records one at a time to determine whether they meet the filter criteria. This generates a lot of traffic, and a system designed without regard to this will leave a user waiting for two minutes for your application to stabilize after reading through 500 records to display the two the user needed. The reality of inefficient client filtering must never be forgotten. Your data should be properly normalized, but analyze it with a view to how it will be *accessed.* If you super-normalize your data, you might cause your application to filter through six tables to find the data to fill that stock item grid. Try to group data in ways that make sense given how it will be retrieved. Optimize your indexes, and carefully plan your master-detail relationships. Consider network traffic issues before adding more referential integrity than you need.

Application presentation issues

If large data downloads are simply unavoidable, then you need to find ways to give the user the illusion of faster progress. For example, you want to steer clear of an application design that loads a multi-megabyte table into a grid when a window is first opened. The user will decide it's taking too long and probably cut off the computer after assuming it has hung. This can lead to data corruption problems on the server. Instead, try to guide the user to what he or she wants via low-overhead lookup tables, and break the data loading process down into multiple parts. Segmenting the load process will work wonders on user perception. For example, instead of requesting a database password from the user before establishing a connection and opening the tables, establish the connection, request the password, and then open the tables. Breaking long processing jobs into smaller ones interspersed with some user action will give the impression that the program is doing something.

Data naming conventions

You'll ease your future upscale project by basing your table and field names on the rules of the platform you hope to move to. Moving your application data from Paradox to Interbase will be easier and will minimize application recoding when your field and table names don't change. Avoid taking advantage of the feel-good conventions of Paradox field naming conventions, such as using spaces within the name. You'll just have to convert them all back when it's time to upscale, which can be a huge synchronization hassle.

Part two: Deployment and coding issues

As with any wide installation of software, you'll want to make installation as trouble-free as possible. It's a given that you'll have to invest in a good BDE-aware installation package such as Great Lakes Business Solution's WISE Installer (http://www.glbs.com). As you'll learn from the Borland Technical Information Documents, there's a BDE parameter that must be set correctly for multi-user Paradox databases: Local Share. It must be set to TRUE in the IDAPI configuration or your clients will frequently lose their changes to the database due to improper flushing of cached updates. Unfortunately, this parameter is set to FALSE by default

in the BDE setup, and the major installation packages won't allow you the option of setting it automatically (they blame it on Borland). My suggestion is to properly script your installation program to alert the user that the parameter must be set and to describe exactly how to do it, even going as far as having the installation package launch the BDE Configuration Manager for the user's convenience. I make it a practice to code the client application to check this parameter upon launching, scolding the user and terminating if this parameter is set incorrectly. There are several freeware components available that allow you to check the value of these parameters at runtime.

Another deployment issue is making sure that the installation procedure properly sets the stage for a successful remote database mapping. Most BDE-aware installation packages allow you to specify aliases to be created, but I recommend setting this programmatically. This mapping is a multi-step process that's often not satisfied by an alias alone. The more of this you control in code, the fewer hours you'll spend troubleshooting problems over the phone with a user 600 miles away.

Step one: Drive mapping

Remote office LANs will have different standards for drive letter mappings, so you can't assume a given drive letter will be free. I recommend programmatically finding a free drive letter instead of allowing the user to choose one. This can be done by dropping an invisible DriveComboBox component on your application's main form. The code in **Listing 1** will use this component and a TStringList to find a free drive.

Listing 1. Finding a free drive letter.

```
{Make sure drive box combo drive letters are
 lower case for search purposes)

 DriveComboBox.TextCase  := tCLowerCase;

{Create StringList of drive letters available}

 DriveList := TStringList.Create;
 DriveList.Assign(DriveComboBox.Items);

(Step through list until drive 'C' is passed)
 Index := 0;
 CurrentDriveLetter  := '';
 MaxxedOut := FALSE;

 while  (CurrentDriveLetter  < 'd') do
    begin
       CurrentDriveLetter  :=
          copy(DriveComboBox.Items[Indexl,  0, 1);
       if Index < DriveComboBox.Items.Count  then
       begin
          Index := Index + 1;
       end
       else
       begin
          WaxxedOut := TRUE;
          break;
       end; (if Index)
    end; {while)

(Increment through alphabet until a gap is
 found.  If the drive list is already full
 (that is,  no drives past 'c'),  set the drive
 letter to 'j'.)

 TestDriveLetter  :=  'd';

 if MaxxedOut = TRUE then (no drives beyond 'd')
 begin
    TestDriveLetter  := 'j';
 end (MaxxedOut = TRUE)
 else
 begin
    while  (TestDriveLetter  =  CurrentDriveLetter)  do
    begin
       TestDriveLetter  := succ(TestDriveLetter);
       Index := Index + 1;
       CurrentDriveLetter  :=
          copy(DriveComboBox.Items[Index   -1], 0, 1);
    end; (while)
 end; (MaxxedOut = FALSE)

(When the above loop finds an unequal drive
 letter,  an unassigned drive has been found.)

 DriveList.Free;
```

Next, create the connection to the NT server database directory share point and map it to the free drive letter. Use dbiGetUserName to pass the user ID to the server (see Listing 2), and the Windows API call WNetAddConnection to perform the mapping.

Listing 2. Perform drive mapping.

```
(Begin setting parameters for network drive login)
(pass the mappable drive letter)
 StrPCopy(lpszLocalName,  TestDriveLetter  +  ':');

(pass the NT server name and share point)
 StrCopy(lpszNetPath,  '\\servername\sharepoint');

(Get user's local WfW, Win95, or Novell login ID, which
you've used to create an account of the same ID on the
host NT server)
 DbiGetNetDserName(userid);

{Prompt user for NT password)
 BadPassword  := FALSE;
 PasswordDialogBox.UserIDLabel.Caption   := userid;
 PasswordDialogBox.ShowModal;
 if (PasswordDialogBox.ModalResult  = mrOK)  then
    begin
       StrPCopy(lpszPassword,
   PasswordDialogBox.PasswordF.ditBox.Text);
    end
 else
    begin
         BadPassword  := TRUE;
    end; (if)

{continue with connection attempt now that all needed
parameters are known)
 if BadPassword  = FALSE then
    begin
       (attempt  network drive mapping)
       result  := WNetAddConnection(lpszNetPath,
IpszPassword,
          1pszLocalName  );

       if (result = 7) then
          begin
             Dialogs.ShowMessage  ('Bad password');
             BadPassword  := TRUE;
          end
       else
```

```
      if result <> WRSUCCESS then
         begin
            MessageDlg('Could not attach drive',
               mtInformation, [mbOk], 0);
            BadPassword := TRUE;
         end;
   end; (if)
```

Step two: Alias and Paradox Net File directory

Set the alias and the Paradox Net File location by adjusting the properties of the TDatabase component and your application's TSession variable, as shown in **Listing** 3. All clients must point to the same PARADOX.NET file; otherwise, record lock tracking won't work and your users will get the cryptic "Multiple .NET Files in Use" BDE error message. Note that the Net file mapping must be identical for all clients. The only allowed difference in the path specification is the drive letter.

Listing 3.Set alias and paradox net directory.

```
with Database1 do
begin
   Close;
   Params.Clear;
   DatabaseName := 'MYTABLES';
   DriverName := 'STANDARD';
   Params.Add('PATH='+ TestDriveLetter + ':\tables');

   {tables is the tail-end of your real path, which is
   ~\\sharepoint\tables". This conforms to Borland's
   suggested directory layout in TI2989)
end; {with)

Session.NetPileDir := TestDriveLetter + ':\netfiles';
Databasel.Connected := True;
```

Step three: Cleaning up

When your program terminates, close your mapped connections with the WnetCancelConnection API call (see **Listing** 4) and reset your IDAPI Configuration parameters.

Listing 4.Closing mapped connections.

```
databasel.close;
Session.DropConnections;
Session.NetFileDir := '';
strPcopy(drive TestDriveLetter + ':');
result := WNetCancelConnection(drive, False);
if result <> WN_SUCCESS then
   begin
      MessageDlg('Could not detach drive ',
         tInformation, [mbOk], 0);
   end;
```

Part three: Deployment and coding issues

As a final deployment issue step, test your entire installation procedure at a remote site and find a willing administrator contact to go through the process with you. Make sure you've properly automated the entire process from installation to successfully loaded tables. Review your setup instructions during the test to make sure they're easy for others to follow. Completely detail all steps.

Conclusion

Carefully following these procedures will get your WAN multi-user application up and running against Paradox. Remember that success is determined by your design planning. You won't have the performance of an expensive client/server system, but you'll end up with a cheap and functional multi-user system.

William Rouck is a consultant working with California Institute of Technology's Jet Propulsion Laboratory. His systems experience includes inventory management and client/server management information systems. wrouck@pop.jpl.nasa.gov.