Realtime publishers

"Leading the Conversation"

The Essentials Series: Optimizing Database Connection Performance

Open Database Connectivity

sponsored by

DataDirect® TECHNOLOGIES

by Mark Scott

Copyright Statement

Open Database Connectivity

Since the SQL Access Group created the Call Level Interface, ODBC has become the most ubiquitous method for connecting to relational database sources (Source: http://support.microsoft.com/kb/110093). ODBC was developed to allow programmers to access relational data in a uniform manner, regardless of the database backend. ODBC translates those generic commands into the specific esoteric commands of the database backend, so the quality of the driver directly determines the performance of the database connectivity layer.

The performance of these drivers varies widely. By understanding the architecture of the ODBC driver, the manner in which it utilizes computer resources, and the scalability of the component, you can make intelligent decisions in choosing and implementing the best drivers for any given enterprise architecture. And with the explosive growth of data access, even small performance gains for each instance can make a big difference in the operation, cost, and maintainability of an organization's data access infrastructure.

Designed to Perform

ODBC is based on a pair of components working at the client application. An ODBC driver manager provides a uniform interface for connecting to and interacting with relational data sources. The architecture is flexible enough that different flavors of SQL can be passed to the server, yet uniform enough that the coder knows what steps he or she must take to work with the database, regardless of what that database might be.

The real magic exists in the ODBC driver. The ODBC driver takes the common commands issued by the developer and translates that to the specific commands used by the target database. This layer of abstraction allows the databases to be developed independent of a constricting standard. They can develop their own individual strengths and new features, without limiting themselves to the restrictions of the standard.

Many database vendors have taken the approach of building ODBC drivers for their databases on top of their database client libraries (see Figure 1), an approach that makes it easier to quickly build ODBC drivers. Driver developers can simply focus on translating ODBC function calls and standard SQL syntax to the client API and database SQL syntax without worrying about coding to the lower-level API of the database itself and managing the headaches of communication over the network.

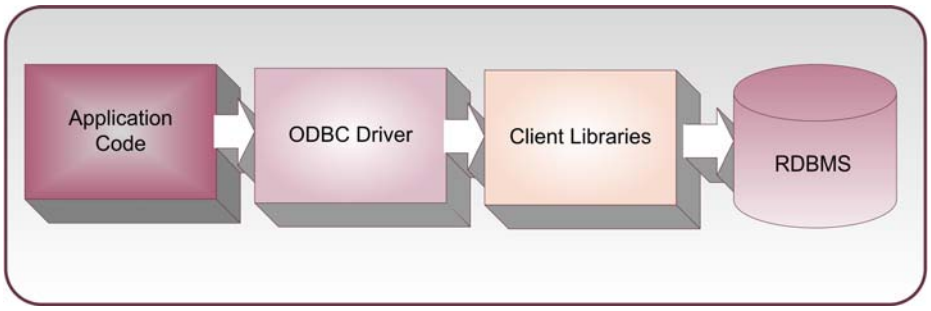

Figure 1: ODBC driver built on top of database client libraries.

An ODBC client that makes direct calls to the database without the use of the client libraries can run more efficiently. By making direct calls to the lower-level interfaces, the ODBC client can expose and leverage the full range of functionality provided by the RDBMS. This allows the data connectivity layer to exploit the full power of the underlying data store. Use of these lower-level calls opens the door for optimization of the database calls. A well-written client speaking more directly to the database will use less memory, operate more efficiently, and provide enhanced facilities for tuning the database calls to the specific workload.

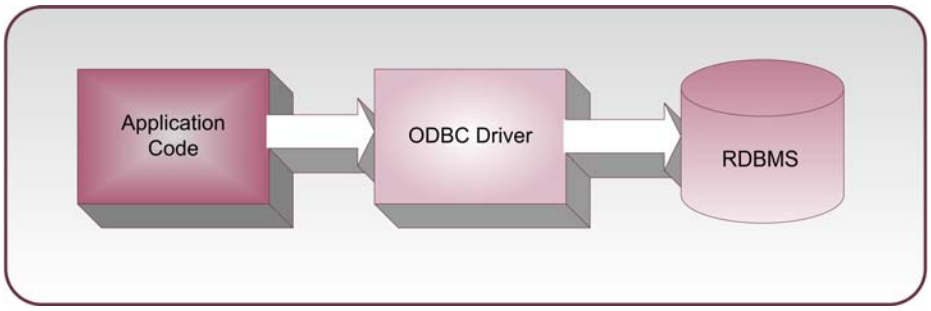

Figure 2: An ODBC client that makes direct calls to the database.

The application platform needs to be considered as well. 64-bit applications provide an extended memory model—very welcome for applications that move large amounts of data such as ETL, data warehousing, and business intelligence applications. The ODBC drivers used in these applications need to run in the same architecture as their client application: x86, x64, IA64, and so on. If the application runs in multiple chip architectures, then multiple versions of the ODBC driver must be utilized.

Some tools restrict full-featured development to a single architecture, such as x86, but ultimately allow the finished product to run in a different architecture. Doing so requires the use of another version of the ODBC driver, one that runs in the architecture in which the final application ultimately runs. Having a complete set of ODBC drivers that offer compatible features across all architectures can simplify development and reduce headaches when the application moves to deployment in production.

Making Efficient Use of Resources

In most enterprises, the initiative is to do more with fewer servers. Power consumption, rack space, and the cost of maintaining multiple servers—each with their own maintenance contract, has caused many organizations to get more from each server. The advent of virtualization technologies makes this even more significant.

Choosing the correct ODBC drivers, drivers that place less resource demands on the server, can make a marked difference in server consolidation projects. To choose the correct drivers, one must examine the real impact of the driver on the system to determine the demand that it generates. This can typically be measured in terms of memory, network, and CPU utilization.

Memory usage should be measured in terms of the entire stack—for example, how much RAM is available before and after all the components are loaded? If an ODBC driver needs to load database client libraries, this will increase the amount of memory consumed. If running a 32-bit application on a 64-bit system requires invoking a different subsystem, that subsystem impacts the OS. The view of system impact needs to be holistic, not limited to simply measuring the size of the binary that one portion of the system opens. This should not be measured by the size of the binaries as they sit on the disk, but rather what actually loads into system RAM during usage.

The impact of RAM consumption can determine how many simultaneous connections a Web server can host, how many processes an enterprise service bus SOA application can manage, and how much parallelism an ETL process can sustain. And if the servers are virtualized, inefficient memory usage can restrict the number of virtual servers that can operate simultaneously on the same physical server.

Network utilization is also important. The overhead for network packets can add up quickly. Connections across WAN links can be performance challenged. As the number of network hops increases, and packets are fragmented, the latency of the connection increases and the performance drops. Different workloads will have different needs. For instance, an ETL process performing a bulk load of the daily transactions through a single connection has different requirements than a Web site that spawns hundreds of connections that move very small data loads. Optimization of the network packets used to service those different loads can have a large impact.

An ODBC driver that can be tuned can help minimize the impact of these workload discrepancies. A driver that can be configured to optimize the lower-level commands it uses to communicate with the database can conduct more efficient, operations with the database.

The CPU is affected by disk I/O, network communications, and memory management. Because these factors interrelate so intimately, real-world workloads are the only accurate test of this effect. Drivers that do well with a small number of connections may falter if the workload is the simultaneous use of hundreds of connections. Understanding how the specific driver behaves, and carefully testing how it performs the type and volume of work that it will encounter in production, will help identify the correct driver for a given use.

It can be difficult to construct realistic tests, but there are many tools to help formulate such tests. The results can be quite revealing and save a lot of re-engineering that might occur if the application is placed into full-scale production and proves ineffective.

Reaching Out Into the Enterprise

The other consideration for the driver is its scalability. There are several issues to address, including resource contention, load balancing, availability, and pooling. For ODBC drivers to work in parallel, they must guard against resource contentions. All software shares dependencies. To use memory pools and common functions, the software components must be able to gain exclusive control of resources for brief periods of time. How well the ODBC driver can orchestrate the process of locking and releasing common resources will determine how many instances can operate concurrently and how well the components can scale on a shared platform.

To scale out, systems often use farms of servers to provide more servers to handle the load. Additionally, products such as Oracle RAC will use load-balanced database servers. Understanding the current and future requirements of the system can help make selection of the appropriate ODBC driver more certain. Research on the load-balancing scenarios that the driver will handle and understanding the options it provides can help reduce maintenance as the system expands.

Database availability is another issue. With differing technologies for implementing high availability, warm standby servers, data mirrors, clusters, grid computing systems, and so on, it is important to know that the ODBC driver can easily and quickly restore system operations when failovers occur. Often, systems that start with low-availability requirements escalate until the enterprise requires them all the time. Using ODBC drivers that complement the availability features and demands of the target database or databases can ease changes to meet evolving requirements.

Pooling is another area where an ODBC driver can improve performance. While balancing the need to minimize resource constrictions, the intelligent use of pools to share common objects, such as connections, can help minimize the total in-memory footprint of the database connectivity software and reduce demands for resources on the servers, both client and database. Effective management of object creation and disposal will also help reduce detrimental effects to the application, such as memory fragmentation or even memory leaks. These factors contribute to the durability and flexibility of how applications can be implemented and how they perform.

Reaching Optimal Performance

The bottom line of any database connectivity product is to minimize the overhead required to send data to and retrieve data from the database. ODBC represents a mature database connectivity technology. That means that it is stable and well understood. It also means that it has gone through the most evolutionary changes, and carries the most baggage from previous iterations. To ensure that the ODBC drivers used in an application are providing the optimal performance, consideration of the driver architecture, resource utilization, and scalability are required.

Ultimately, the architecture of the ODBC driver will lay the foundation for the resource utilization and scalability. ODBC drivers that communicate directly to the database without loading additional libraries have the potential of being the most efficient. They can reduce the total memory footprint and open the door for optimization. Drivers that operate in the required mode (32-bit versus 64-bit) and can operate natively in process with the clients can provide improved performance.

Resource utilization needs to be considered holistically. The overall memory footprint, network bandwidth consumption, and CPU cycle utilization combine to put a load on the server. Seeing this impact in context with real-world workloads will determine the cost of the database connectivity, expressed in terms of how large server systems need to be, or how many of them are required. Efficient systems can help server consolidation and virtualization projects accelerate.

Systems grow over time. The ability to scale out systems can take many forms. Placing additional requirements on existing servers, scaling out through farms and grids, and enhancing availability all have an impact on how well the systems meet enterprise service level agreements (SLAs). Choosing the correct ODBC driver up front and tuning it to the needs of the actual workload presented can help IT deliver on those commitments.