Recommendations and Techniques for

Scaling Microsoft SQL Server

To support many more users, a database must easily scale out as well as up. This article describes techniques and strategies for scaling out the Microsoft® SQL Server relational database management system (RDBMS) and provides scenarios illustrating scale-out deployments.

BY DON JONES

Most enterprise applications today run on a Microsoft® Windows®, UNIX®, or Linux® operating system–based relational database management system (RDBMS), such as Microsoft SQL Server 2000. Scalability has become a critical factor in the success of these applications as the number of users relying on them has grown.

The Internet also has profoundly affected the need for scalability. Once exposed to just a few thousand users, the data in many corporate databases now must be accessed by tens of thousands of concurrent users through e-commerce sites, Web services, and other Internet-based applications. Scaling databases to support these users is a major concern for both database software developers and database administrators.

Differences between scaling up and scaling out

When database performance worsens, administrators typically address the problem first by scaling up—that is, by trying to optimize performance in the current environment. Because many database applications have inefficient designs or become inefficient as their usage patterns change, finding and improving the areas of inefficiency can yield significant performance benefits. Fine-tuning the database server can help perform more queries, handle more users, and run more efficiently.

SQL Server scales up fairly well—to a point. In one real-world scenario, for example, a company's database required a nine-table join to look up a single customer address. Selectively denormalizing the tables and applying strategic indexes allowed SQL Server to execute address queries much faster. Because address lookups were a common task for this company, even a minor per-query improvement significantly enhanced overall server performance.

Unfortunately, scaling up is limited in how much it can improve an application's performance and ability to support more users. For example, take a database whose sole function is to perform a single, simple query—no joins, no need for indexes. A high-performance SQL Server computer—for example, a quad-processor server with 4 GB of RAM and

When database performance worsens, administrators typically address the problem first by scaling up.

several fast hard drives—could probably support tens of thousands of users who must concurrently execute that one query. However, this server might not be able to support a million users. In this situation, scaling up—fine-tuning—would be insufficient, because such a simple query leaves little room for improvement. To begin supporting many more users, scaling out is a better solution.

Scale-out strategies redistribute workloads

Scaling out SQL Server, a more complicated process than scaling up, requires splitting a database into various pieces, then moving the pieces to different, independent SQL Server computers. The grocery-store checkout line presents a good analogy for comparing the two processes. In a busy grocery store with only one checkout lane open, a long line of unhappy customers would quickly materialize.

A scale-up approach—installing faster barcode scanners, requiring everyone to use a credit card instead of writing a check, or hiring a faster cashier—can make the checkout process itself more efficient. These measures might improve the situation, but not solve the problem; customers would move through the line more quickly, but they still would have only one checkout lane.

A better solution would be to scale out—in this analogy, by opening additional checkout lanes. Customers could now be processed in parallel by completely independent lanes. To make the analogy closer to a database scale-out scenario, the grocery store could have specialized lanes: one that expedites processing (customers purchasing 15 items or fewer), and another that focuses on produce, which often takes longer because it must be weighed and not simply scanned.

An ideal, if unrealistic, solution might be to retain a single lane for each customer, but to divide each customer's purchases into categories to be handled by specialists: produce, meat, boxed items, and so forth. Specialized cashiers could minimize their interactions with each other, keeping the process moving speedily along. Although unworkable in a real grocery store, this solution illustrates a real-world model for scaling out databases.

General strategies for scaling out databases

Database managers can consider two basic scale-out strategies for distributing the workload of a database across multiple servers. Most major RDBMS platforms, including SQL Server, provide the means to make these strategies possible.

SQL Server farms replicate the database

The first approach simply adds more servers. Consider a scenario in which a company has an office in New York and one in Los Angeles. Both offices have several thousand users who frequently query data from a corporate application, such as an order-processing database. Users rarely change data in the system, but they frequently add new data. In this scenario, users in both offices are overloading the database. Even if the database is a well-written multitier application, processing all the information on only one database server at the back end can create a bottleneck.

Scaling out SQL Server, a more complicated process than scaling up, requires splitting a database into various pieces, then moving the pieces to different, independent SQL Server computers.

Figure 1 illustrates one way to address the problem: a SQL Server farm. In this technique, two database servers each contain a complete copy of the database. Each office houses one server, and the users in each office connect only to their local server. Changes and new records are replicated between the servers by using SQL Server replication. To avoid conflicts when adding new records, each office might, for example, be assigned a unique range of order ID numbers, ensuring that new records created in either office can be uniquely identified across both copies of the database.

This strategy is perhaps the simplest means of scaling out SQL Server. Although replication is not easy to set up and maintain on SQL Server, neither is it extremely difficult. The strategy works well even with many servers and copies of the database.

However, the data replication strategy does incur some drawbacks, especially latency. Neither copy of the database will ever match the other exactly. As new records are added to each copy, time elapses before replication begins. With only two servers in the company, each server might be as much as an hour out of sync with the other, depending upon how administrators set up replication.

Adding more servers, however, involves difficult replication decisions. For another scenario, consider the six-office setup depicted in Figure 2. Each of the six offices has its own independent SQL Server system—an excellent design for scalability. However, latency could be very high. If each SQL Server replicates with its partners just once every hour, then total system latency could be three hours or more. A change made in the Los Angeles office would replicate

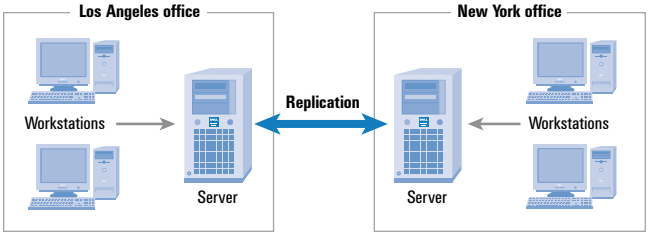

Figure 1. SQL Server farm

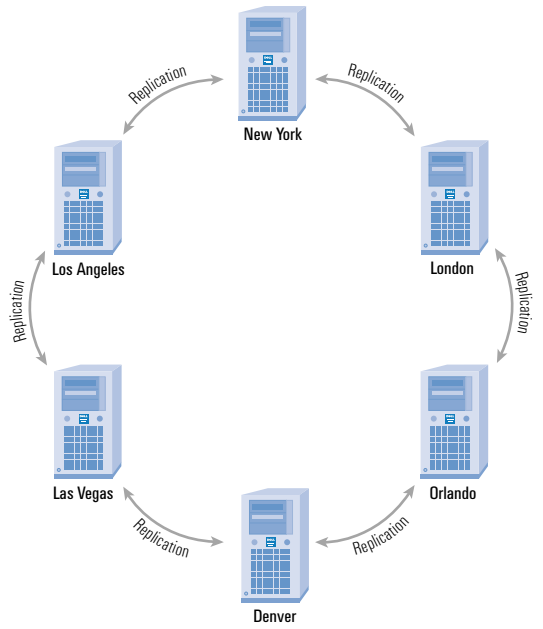

Figure 2. Six-server farm

to New York and Las Vegas in about an hour. An hour later, the change would reach London and Denver. An hour later, it would arrive in Orlando. Given such high latency, the entire system would probably never be synchronized completely.

Administrators can reduce latency, but at a performance cost. If each of the six servers replicated with each of the other five servers, the system could *converge,* or be universally in sync, about once an hour (assuming again that replication occurred every hour). Figure 3 shows such a *fully enmeshed* design.

In this fully enmeshed design, each server must maintain replication agreements with five other servers, and must replicate with each server every hour. This much replication, particularly in a busy database application, would likely slow response so much that the performance gain achieved by creating a server farm would be lost. Each office might require two servers just to maintain replication and meet users' needs. Although fairly easy to implement, the server farm technique has a point of diminishing returns.

Distributed partitioned databases move tasks to different servers

A more sophisticated strategy—but one that is also more difficult to implement—involves partitioning the database and moving the pieces to different servers. Unlike the simplified order-processing database example previously discussed in "SQL Server farms replicate the database," most real-world database applications tend to rely on an equal mix of data reading and data writing. For example, an order-processing application might include a product catalog that is largely read only, a customer-order database that is write heavy, and tables containing supplier information that are equally read-write.

Scaling out SQL Server can offer benefits not only in improved application performance, but also in greater redundancy and availability.

These three closely related database segments—catalog, orders, and supplier tables—are fairly task-independent: diverse users within the organization tend to use each database differently. Merchandisers might write to the catalog but do little else. Customer service representatives might read the catalog and write to the orders tables but never access the supplier tables. The warehouse staff might read the catalog and read from and write to the supplier tables. This division of labor indicates where the database itself can be split, as Figure 4 illustrates.

Administrators can use two basic approaches to implementing the distributed partitioned database strategy. The first is to modify the client application so that it understands the division of the database across multiple servers. Straightforward yet somewhat time-consuming, this solution does not work well for the long term. Future changes to the application could result in additional divisions, which would in turn require additional reprogramming.

A better approach is to program the client application to use stored procedures, views, and other server-side objects—an ordinary best practice for a client-server application—so that the client application need not be aware of the physical location of the data. SQL Server offers different techniques, such as distributed partitioned views, to handle this setup.

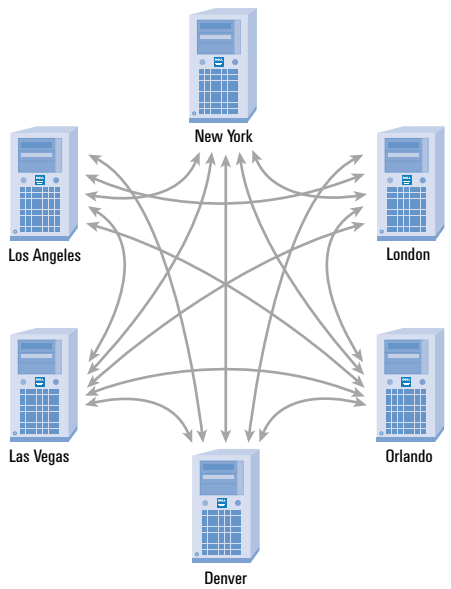

Figure 3. Fully enmeshed six-server farm

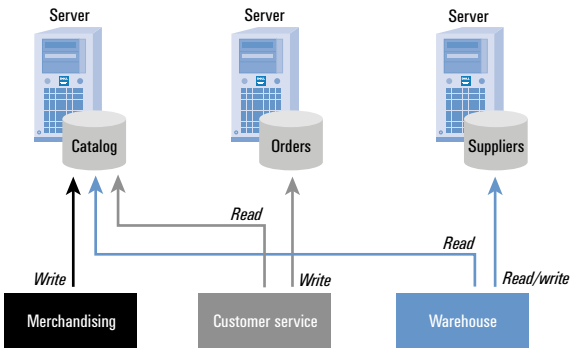

Figure 4. Identifying task-based divisions in the database design

Scale-out techniques using SQL Server and Windows

SQL Server and Windows offer several techniques to enable scaling out, including SQL Server–specific features such as distributed databases and views and Windows-specific functions such as Windows Clustering.

Distributed partitioned views help create virtual tables

SQL Server distributed partitioned views allow developers to create views that combine tables from multiple SQL Server computers into a single virtual table. This method logically divides a database across multiple SQL Server computers. Rather than reprogramming client applications to understand the division of the databases, developers can create distributed views that present a virtualized version of them. These tables appear to client applications as if they were on a single server. Meanwhile, SQL Server combines the tables, which are spread across multiple servers, into a single view.

Distributed views are a powerful tool in scaling out. They allow developers to redistribute databases transparently to the end users and their business applications. As long as client applications are designed to use the views rather than the direct tables, the tables themselves can be rearranged and scaled out as necessary without the client application being aware of any change.

SQL Server and Windows offer several techniques to enable scaling out, including SQL Server–specific features such as distributed databases and views and Windows-specific functions such as clustering.

The workload required to create and present the view to client computers is shared by all servers participating in the view—or by all servers in the federation. SQL Server 2000 is the first version of SQL Server to make a significant improvement to this approach, because the data within the views can be updated by client applications as if the data were in a regular table. The updates are cascaded back to the necessary participant servers.

Replication of distributed partitioned databases reduces latency

Another scale-out approach involves partitioning a database across multiple servers and then replicating the database copies. Like the six-server order-processing farm described earlier, each server contains a complete database. In this method, each server is responsible for a different set of rows. SQL Server replication is used to keep each copy of the database updated. This method allows each server to immediately access its own rows and provides reasonably low latency for access to rows created on other servers. Client applications often must be modified to understand this structure. In many partitioned database schemes, data rows may be modified only on the server that owns them, with the changes then being moved to the other servers through replication. Client applications must know how to determine which server owns a row before making modifications.

Windows Clustering facilitates high availability and scalability

Besides improving performance, Windows Clustering can help avoid the risk of server failure when scaling out. For example, a two-node active/active cluster has two independent SQL Server servers. These nodes can be configured as a server farm, in which each server contains a complete copy of the database and users are distributed between them. An alternative is a distributed database architecture, in which each server contains one logical half of the entire database. In either architecture, a failure of one server is not catastrophic because Windows Clustering enables the other server to transparently take over and act as two servers.

Over-engineering is the key to a successful active/active cluster. Each node should be designed to operate at a maximum of 60 percent capacity. If one node fails, the other node can begin running at 100 percent capacity, incurring only about a 20 percent loss of efficiency. Still, performance is generally well within an acceptable range considering that, after failover, applications must run on half as much hardware.

Setting up clusters can be extremely complex. In Windows Clustering, the software is not difficult to use, but the underlying hardware must be absolutely compatible with Windows Clustering—and most hardware vendors have exacting requirements for cluster setups. Purchasing preconfigured clusters from a major server vendor, such as Dell, can help simplify cluster setup. The cluster is designed to be ready to run on delivery, and both the vendor and Microsoft can provide cluster-specific technical support if necessary.

High-performance storage to boost SQL Server response

High-performance storage is an often-overlooked performance benefit for SQL Server—particularly external storage area networks

UNDERSTANDING DATABASE INEFFICIENCY

Databases can be inefficient for several reasons:

- **Poor design:** Many application developers do not excel at database design. Some, for example, have been taught to fully normalize the database at all costs, which can lead to significantly degraded performance. Sometimes project schedules do not permit enough design iterations before the database must be locked down and software development begins. In some cases, the application itself is not designed well, resulting in an incomplete database design that must be patched and expanded as the application is created.
- **Change:** An application used in a way unintended by its designers can reduce efficiency. The application may have expanded and begun suffering from "scope creep"—the growth or change of project requirements. In this case, redesigning the application from the beginning to meet current business needs may be the best solution to database inefficiency.
- **Growth:** Databases are designed for a specific data volume; once that volume is exceeded, queries may not work as they were intended. Indexes might need to be redesigned or at least rebuilt. Queries that were intended to return a few dozen rows may now return thousands, affecting the underlying design of the application and the way data is handled.

These problems are difficult to address in a live, production application. Scaling up tends to have a limited effect. Although developers may agree that the application's design is inefficient, companies are reluctant to destroy a serviceable application and start over without serious consideration.

Scaling out can offer a less drastic solution. Although scaling out requires considerable work on the server side, it may not require much more than minor revisions to client-side code, making the project approachable without completely re-architecting the application. Scaling out might not be the most elegant or efficient way to improve performance, but it does help alleviate many database and application design flaws. It also can allow companies to grow their database applications without needing to redesign them from the beginning.

(SANs) that rely on Fibre Channel technology rather than traditional SCSI disk subsystems. Because high-performance storage enables an existing server to handle a greater workload, it constitutes an example of scaling up rather than out.

SQL Server is a highly disk-intensive application. Although SQL Server includes effective memory-based caching techniques to reduce disk reads and writes, database operations require significant data traffic between a server's disks and its memory. The more quickly the disk subsystem can move data, the faster SQL Server will perform. Some industry estimates suggest that 75 percent of idle time in SQL Server results from waiting for the disk subsystem to deliver data. Improving the speed of the disk subsystem can markedly improve overall SQL Server performance.

Moving to additional RAID-5 arrays on traditional copper SCSI connections is a simple way to improve disk space. However, high-speed Fibre Channel SANs offer the best speed, as well as myriad innovative recovery and redundancy options—making them a safer place to store enterprise data.

Scale-out strategy for improving SQL Server performance, redundancy, and availability

As applications grow to support tens and hundreds of thousands of users, scaling is becoming a mission-critical activity. Scaling up—improving efficiency by fine-tuning queries, indexes, and so forth—helps IT organizations do more with less. However, scaling up can require high administrative overhead and may have limited effect. Administrators might spend two weeks to achieve a 1 percent performance gain, an improvement that cannot compare to the much higher gains promised by a well-planned scale-out design.

Although seldom considered as a target for scaling out, SQL Server is well suited to this strategy, in both server farm and more sophisticated distributed database approaches. Scaling out SQL Server can offer benefits not only in improved application performance, but also in greater redundancy and availability.

Don Jones is a founding partner of BrainCore.Net, and has more than a decade of experience in the IT industry. Don's current focus is on high-end enterprise planning, including data availability and security design.

FOR MORE INFORMATION

This article is based on an excerpt from the free eBook *The Definitive Guide to Scaling Out SQL Server* (Realtimepublishers.com) by Don Jones, available at http://www.dell.com/sql/ebook

Dell | Microsoft SQL Server 2000: http://www.dell.com/us/en/esg/topics/products_software_pedge_001_database.htm

Microsoft SQL Server: http://www.microsoft.com/sql