realtimepublishers.com™

The Definitive Guide™ To

Scaling Out SQL Server 2005

Don Jones

Introduction to Realtimepublishers

by Sean Daily, Series Editor

The book you are about to enjoy represents an entirely new modality of publishing and a major first in the industry. The founding concept behind Realtimepublishers.com is the idea of providing readers with high-quality books about today's most critical technology topics—at no cost to the reader. Although this feat may sound difficult to achieve, it is made possible through the vision and generosity of a corporate sponsor who agrees to bear the book's production expenses and host the book on its Web site for the benefit of its Web site visitors.

It should be pointed out that the free nature of these publications does not in any way diminish their quality. Without reservation, I can tell you that the book that you're now reading is the equivalent of any similar printed book you might find at your local bookstore—with the notable exception that it won't cost you $30 to $80. The Realtimepublishers publishing model also provides other significant benefits. For example, the electronic nature of this book makes activities such as chapter updates and additions or the release of a new edition possible in a far shorter timeframe than is the case with conventional printed books. Because we publish our titles in "real-time"—that is, as chapters are written or revised by the author—you benefit from receiving the information immediately rather than having to wait months or years to receive a complete product.

Finally, I'd like to note that our books are by no means paid advertisements for the sponsor. Realtimepublishers is an independent publishing company and maintains, by written agreement with the sponsor, 100 percent editorial control over the content of our titles. It is my opinion that this system of content delivery not only is of immeasurable value to readers but also will hold a significant place in the future of publishing.

As the founder of Realtimepublishers, my *raison d'être* is to create "dream team" projects—that is, to locate and work only with the industry's leading authors and sponsors, and publish books that help readers do their everyday jobs. To that end, I encourage and welcome your feedback on this or any other book in the Realtimepublishers.com series. If you would like to submit a comment, question, or suggestion, please send an email to feedback@realtimepublishers.com, leave feedback on our Web site at http://www.realtimepublishers.com, or call us at 800-509-0532 ext. 110.

Thanks for reading, and enjoy!

Sean Daily
Founder & Series Editor
Realtimepublishers.com, Inc.

Copyright Statement

Chapter 1: An Introduction to Scaling Out

Enterprise applications have become more complex and have taken on a greater burden for managing a company's critical data. At the same time, the amount of data managed by those applications has swelled exponentially as companies begin to track an increasingly greater amount of information—data about customers, vendors, sales, and more. In addition, the advent and popularity of data warehousing has expanded the amount of stored data by a factor of a hundred or more. In short, we're keeping track of more data than ever before, and our database servers are starting to show signs of strain. This strain has been the catalyst for interest in scaling out from a diverse audience that ranges from database administrators to CEOs. Even your organization's CFO will care, as scaling up is often more expensive than scaling out, especially when scaling up requires the purchase of an expensive midrange or mainframe platform.

What Is Scaling Out?

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, as well as moving to more powerful hardware or upgrading existing hardware—helps IT organizations do more with less. In the past, scaling up met the increasing IT needs of many organizations. After all, even an older Windows NT 3.51 server can address up to 4GB of physical RAM. Newer 64-bit machines can address terabytes of physical RAM, which, even by today's standards, seems practically infinite. And yet, thanks to the incredible amount of data that organizations handle, all that hardware isn't always sufficient.

Scaling out is the process of making multiple servers perform the work of one logical server or of dividing an application across multiple servers. A Web farm provides an excellent example of scaling out (see Figure 1.1). Each server in the farm is completely independent and hosts an identical copy of the entire Web site. Users are load balanced across the servers, although the users rarely realize that more than one server exists. For example, when you visit http://www.microsoft.com, can you tell how many Web servers are actually behind the scenes handling your request? The number is greater than one or two, you can be sure.

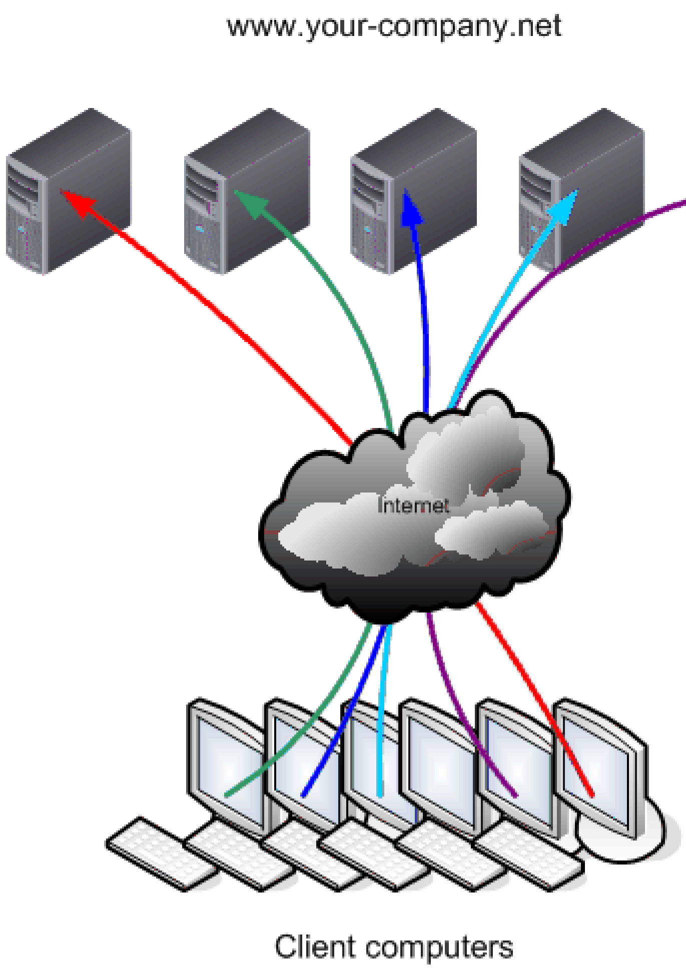

Figure 1.1: Web farms represent an excellent example of scaling out.

Why is scaling out even necessary in a Web farm? After all, most Web servers are fairly inexpensive machines that employ one or two processors and perhaps 2GB of RAM or so. Surely one really pumped-up server could handle the work of three or four smaller ones for about the same price? The reason is that individual servers can handle only a finite number of incoming connections. Once a server is dealing with a couple of thousand Web requests (more or less, depending on the server operating system—OS, hardware, and so forth), adding more RAM, network adapters, or processors doesn't increase the server's capacity enough to meet the demand. There is always a performance bottleneck, and it's generally the server's internal IO busses that move data between the disks, processors, and memory. That bottleneck is literally built-in to the motherboard, leaving you few options other than scaling out. Figure 1.2 illustrates the bottleneck, and how adding processors, disks, or memory—the features you can generally upgrade easily—can only address the problem up to a point.

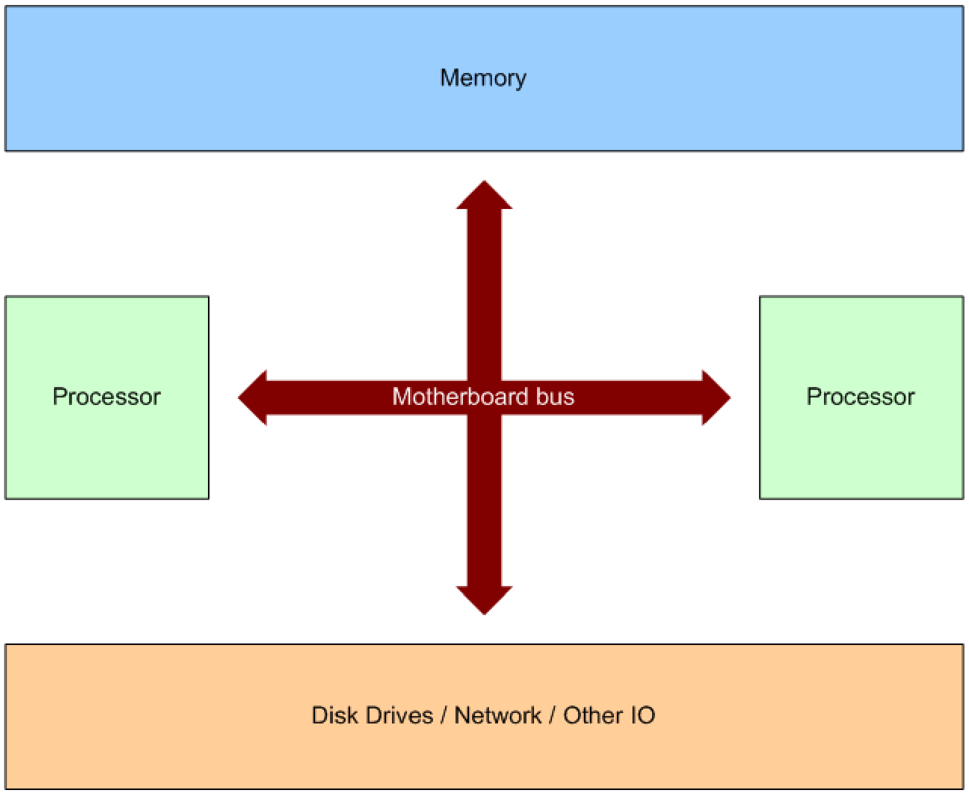

Figure 1.2: The motherboard bottleneck.

Why Scale Out Databases?

Database servers don't, at first glance, seem to share a lot of characteristics with Web servers. Web servers tend to deal with small amounts of data; database servers must handle thousands of times more information. Database servers are usually "big iron" machines with multiple processors, tons of RAM, fast disks, and so forth—they certainly don't suffer from a shortage of raw computing power. In addition, database servers don't always accept connections directly from clients; multi-tier applications provide one or more intermediate layers to help consolidate connections and conserve database servers' capacity.

Database servers also have vastly more complicated jobs than Web servers. Web servers read files from disk and stream the files out over the Internet; database servers do an incredible amount of work—even adding a single row to a small database table might require a half-dozen indexes to be updated, triggers to be executed, other connections to be notified of the new row, and so forth. The work of a database server eventually results in a cascade of additional work.

Database servers tend to bottleneck at several levels. Although 4GB of RAM is a lot of memory, a database server might be dealing with 8GB worth of data changes. Four processors sounds like a lot—and it is—but a database server might be asking those processors to deal with cascading changes from thousands of database operations each minute. Eventually those changes are going to have to be dumped to disk, and the disks can only write data so fast. At some point, the hardware isn't going to be able to keep up with the amount of work required of database servers. Thus, scaling out is becoming critical for companies that rely on truly large databases.

What Is a Truly Large Database?

For a view of a truly large database, take a look at Terraserver.com. This system contains satellite imagery down to 1 or 2 meters of detail of nearly the entire world. This amount of data is literally *terabytes* of information; so much so that simply backing up the information requires high-powered hardware. Terraserver.com does more than simply store and query this vast amount of data, which is primarily the photographs stored as binary data; it is Internet-accessible and has handled tens of thousands of user connections per hour.

Making this size of database available to a large audience of users wouldn't be possible without scaling out. In fact, Terraserver.com's traffic has been high enough to take down Internet Service Providers (ISPs) whose customers were all hitting the Terraserver.com site—particularly when the site released photos of Area 51 in Nevada. Although the ISPs had problems, Terraserver.com never went down during this period. Other giant databases are easy to find. Microsoft's Hotmail Web site, for example, deals with a good bit of data, as does Yahoo!'s email service. America Online has a pretty huge database that handles its membership records for tens of millions of users. Big databases are everywhere.

Microsoft SQL Server and Scaling Out

Most organizations tackle database performance issues by scaling up rather than out. Scaling up is certainly easier, and Microsoft SQL Server lends itself very well to that approach; however, scaling up might not always be the best choice. Many database applications have inefficient designs or grow inefficient over time as their usage patterns change. Simply finding and improving the areas of inefficiency can result in major performance benefits. For example, I once worked with a customer whose database required a nine-table join to look up a single customer address. Selectively denormalizing the tables and applying appropriate indexes let SQL Server execute customer address queries much faster. As address lookups were a very common task, even a minor per-query improvement resulted in a major overall performance benefit.

Inefficiency: It's Nobody's Fault

Why are databases inefficient? Determining a single culprit is a difficult task as there are several reasons why a database's performance might be less than stellar. Heading the list is poor design, which can result from a number of sources. First, 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 is not designed well, resulting in an incomplete database design that must be patched and expanded as the application is created.

Change is another major factor. An application might be used in a way unintended by its designers, which reduces efficiency. The application may have expanded and begun suffering from *scope creep*—the growth of change of project requirements. In this case, redesigning the application from the beginning to meet current business needs might be the best solution to database inefficiency.

Sheer growth can also be a problem. Databases are designed for a specific data volume; once that volume is exceeded, queries might 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 might now return thousands of rows, 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—trying to optimize performance in the current environment—tends to have a limited effect. Nobody will debate that the application's design is inefficient, but companies are reluctant to destroy a serviceable application without serious consideration. Scaling out provides a less-drastic solution. Although scaling out requires much work on the server side, it might not require much more than minor revisions to client-side code, making the project approachable without completely re-architecting the application. Although scaling out might not be the most elegant or efficient way to improve performance, it helps alleviate many database and application design flaws, and can allow companies to grow their database applications without needing to redesign them from scratch.

There is a limit to how much scaling up can improve an application's performance and ability to support more users. To take a simplified example, suppose you have a database that has the sole function of performing a single, simple query. No joins, no real need for indexes, just a simple, straightforward query. A beefy SQL Server computer—say, a quad-processor with 4GB of RAM and many fast hard drives—could probably support tens of thousands of users who needed to simultaneously execute that one query. However, if the server needed to support a million users, it might not be able to manage the load. Scaling *up* wouldn't improve the situation because the simple query is as finely tuned as possible—you've reached the ceiling of scaling up, and it's time to turn to scaling *out*.

Scaling out is a much more complicated process than scaling up, and essentially requires you to split a database into various pieces, then move the various pieces to independent SQL Server computers. Grocery stores provide a good analogy for comparing *scale up* and *scale out*. Suppose you drop by your local supermarket and load up a basket with picnic supplies for the coming holiday weekend. Naturally, everyone else in town had the same idea, so the store's a bit crowded. Suppose, too, that the store has only a single checkout lane open. Very quickly, a line of unhappy customers and their shopping carts would stretch to the back of the store.

One solution is to improve the efficiency of the checkout process: install faster barcode scanners, require everyone to use a credit card instead of writing a check, and hire a cashier with the fastest fingers in the world. These measures would doubtless improve conditions, but they wouldn't solve the problem. Customers would move through the line at a faster rate, but there is still only the one line.

A better solution is to *scale out* by opening additional checkout lanes. Customers could now be processed in parallel by completely independent lanes. In a true database scale-out scenario, you might have one lane for customers purchasing 15 items or less, because that lane could focus on the "low-hanging fruit" and process those customers quickly. Another lane might focus on produce, which often takes longer to process because it has to be weighed in addition to being 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.

📖 In Chapter 2, we'll explore scale up versus scale out in more detail. Scaling up is often a prerequisite for scaling out, so I'll also provide some tips and best practices for fine-tuning your databases into scaled-up (or scale-out-able) shape.

Scaling out is generally appropriate for only very large databases. Consider scaling up for fairly small databases that warrant scaling out to improve performance. Improve the hardware on which the database is running, improve the procedures and queries used to manage the database, or correct fundamental database design flaws. Scaling out isn't a last resort, but it is best reserved for well-designed databases being accessed by a large number of well-designed clients.

SQL Server 2005 Editions

Of course, not every edition of SQL Server is appropriate for scaling out. SQL Server 2005, for example, comes in an Express and a Workgroup edition, neither of which are appropriate to scale-out scenarios. Both of these editions are designed for smaller databases with a very small number of users. SQL Server 2005 Standard Edition and SQL Server 2005 Enterprise Edition (particularly the latter) feature the functionality and features that make scale-out a reality. It's important to understand that, while *all* editions of SQL Server 2005 share a substantially similar core feature set, this book will focus on the Standard and Enterprise editions of the product. Although you might be able to implement many of the techniques I'll describe using the Express or Workgroup editions, don't expect those editions to match the performance of the Standard and Enterprise editions.

General Technologies

There are several core technologies, most of which are common to all high-end database platforms, that make scale-out possible. These technologies include:

- Data access abstraction—Often implemented as views and stored procedures, these abstractions allow client and middle-tier applications to access data in a uniform fashion without an understanding of how the data is actually stored on the back-end. This allows a client application to, for example, query data from a single view without realizing that the data is being assembled from multiple servers.
- Replication—A suite of technologies that allows multiple read or read-write copies of the same data to exist, and for those copies to undergo a constant synchronization process that seeks to minimize the time during which the copies (or *replicas*) are different from one another..
- Clustering—A set of technologies that allows multiple servers running complex applications (such as SQL Server) to appear as a single server. In some cases, the servers distribute the incoming workload among themselves; in other scenarios, the servers act as backups to one another, picking up workload when a single server fails.

Clustering is an overused, overloaded term; in this context, I'm referring to clustering in a general sense. *Windows Clustering* is a specific technology that provides high availability. Although it is not in and of itself a scale-out technology (it does nothing to distribute workload), it does have a useful place in helping scaled-out solutions become more reliable and more highly available.

- Server hardware—The brawn behind the brains, server hardware—memory, processors, storage devices, and so forth—provides the power to run complex applications like SQL Server. Advances in server hardware, such as faster bus and processor speeds, more efficient memory designs, and new-generation processors (such as 64-bit) provide additional capability, allowing individual servers to handle ever-greater workloads. So-called *commodity hardware* is built around industry-standard designs, and spreads the cost of researching and developing the hardware across literally millions of customers. The result is extremely powerful server hardware that is both affordable and easier to support.

All of these technologies play a role in scale-out solutions, and I'll be discussing them in more detail throughout this book.

General Scale-Out Strategies

When the time comes to scale out, what sort of strategy should you take? There are a couple of general strategies you should consider. These strategies aren't technology specific; they're simply general ways to distribute the workload of a database across multiple servers. SQL Server—as well as most major relational database management system (RDBMS) platforms—provides technologies to make these strategies possible.

SQL Server Farms

The first approach is to simply put more servers on the job. For example, suppose your 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. Changes to the data are fairly rare, although new data is frequently added. Order-processing databases are an excellent example of this type of scenario: new orders are added quickly, but existing orders don't really change once they're in the system. Your problem in this case is that the database is being slammed by users in both offices. Although you built a useful multi-tier application, the bottleneck results from the fact that only one database server is handling the back end.

Figure 1.3 illustrates one solution to this problem: a SQL Server farm in which two database servers each contain a complete copy of the database. One server resides in each office, 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 is 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.

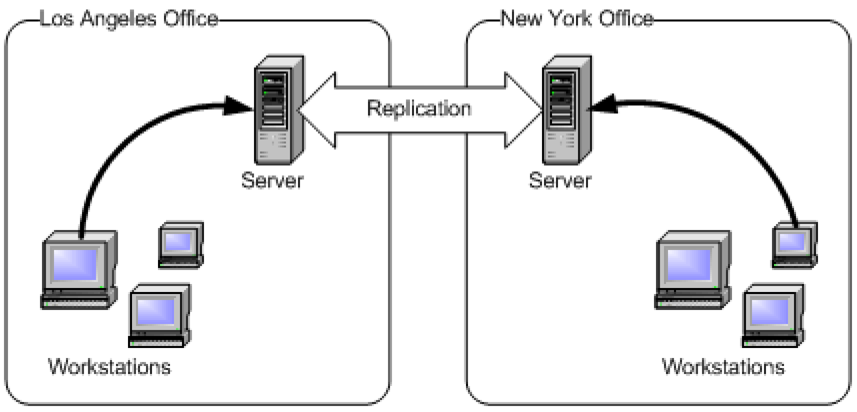

Figure 1.3: An example SQL Server farm.

This strategy is perhaps the simplest means of scaling out SQL Server. Although SQL Server replication isn't simple to set up and maintain, the strategy works well even with many different servers and copies of the database. However, this setup has drawbacks. *Latency* is the primary drawback—neither copy of the database will ever be exactly like the other copies. As new records are added to each copy, a period of 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 you set up replication. Adding more servers, however, involves difficult replication decisions. For example, consider the six-office setup that Figure 1.4 depicts.

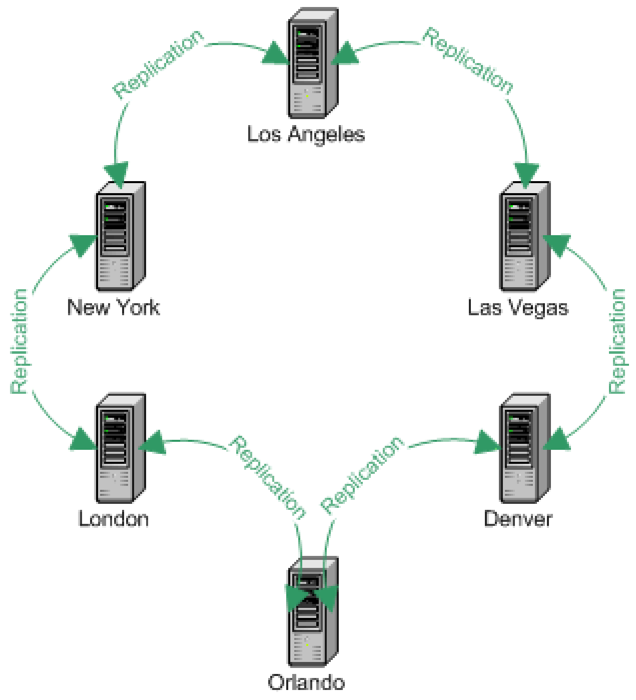

Figure 1.4: A six-server farm.

In this example, each of the six offices has an independent SQL Server, which is a useful and scalable design. However, latency might be very high. If each SQL Server replicated with its partners just once every hour, then total system latency could be 3 hours or more. For example, a change made in the Los Angeles office would replicate to New York and Las Vegas in about an hour. An hour later, the change would make it to London and Denver. Another hour later, and the change would finally reach Orlando. With such high latency, it's unlikely that the entire system would ever be totally in sync.

Latency can be reduced, but at the cost of performance. For example, if each of the six servers replicated with each of the other six servers, the system could *converge*, or be universally in sync, about once an hour (assuming again that replication occurred every hour). Figure 1.5 shows this *fully enmeshed* design.

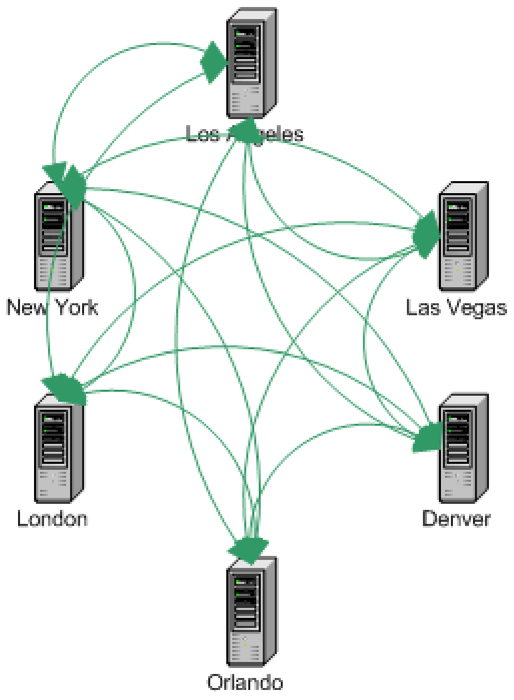

Figure 1.5: A fully enmeshed six-server farm (the green lines represent replication).

The consequence of this design is decreased performance. Each server must maintain replication agreements with five other servers and must perform replication with each other server every hour. So much replication, particularly in a busy database application, would likely slow performance 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. Therefore, the server farm technique, although fairly easy to implement, has a point of diminishing return.

Distributed Partitioned Databases

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, most real-world database applications (aside from business intelligence databases) 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 write-heavy customer-order database, and tables containing supplier information that are equally read-write. These three database segments—catalog, orders, and supplier tables—although closely related, are fairly task-independent: different 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 can be split, as Figure 1.6 illustrates.

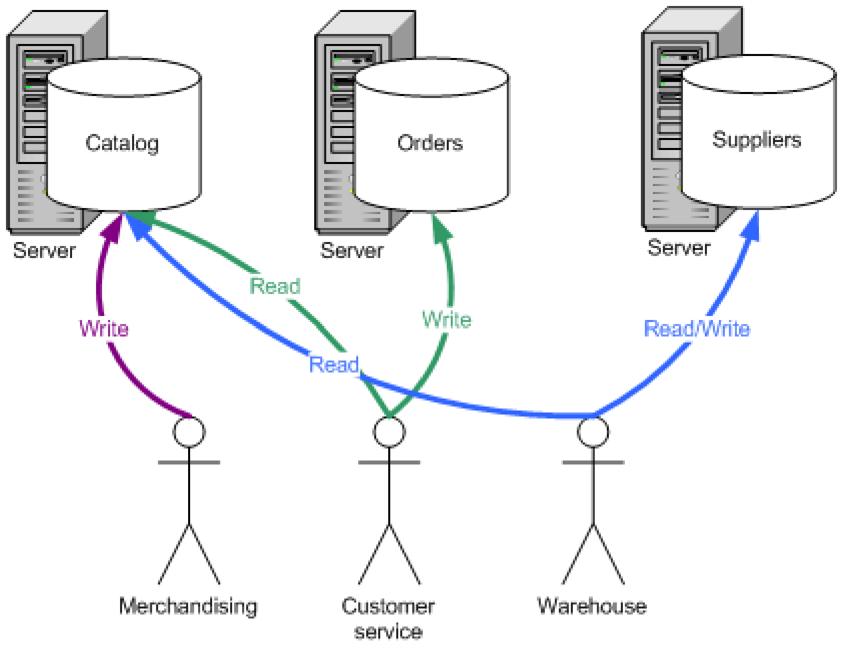

Figure 1.6: Identifying task-based divisions in the database design.

Administrators can use two basic approaches to implement this strategy. The first approach is to modify the client application so that it understands the division of the database across multiple servers. Although fairly straightforward, if 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 does not need to be aware of the data's physical location. SQL Server offers different techniques to handle this setup, including distributed partitioned views.

Scale-Out Techniques

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 (which, as I've mentioned, isn't specifically a scale-out technology although it does have a use in scale-out scenarios).

Distributed Partitioned Views

SQL Server views allow you 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, you can create views that present a virtualized version of those tables. The tables appear to client applications as if the tables were on a single server. Meanwhile, SQL Server combines the tables, which are spread across multiple servers, into a single view.

Views are a powerful tool in scaling out. They allow you 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.

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 this approach significantly useful, because the data within the views can be updated by client applications as if the data were a regular table; the updates are cascaded back to the necessary participant servers.

Distributed partitioned views are a significantly enhanced version of views. Distributed partitioned views enable you to horizontally partition tables so that several servers each contain different rows from the table. The distributed partitioned view is stored on all the servers involved, and combines the rows from each server to create a single, virtual table that contains all the data. Figure 1.7 illustrates data from three servers brought together into one view for a client application.

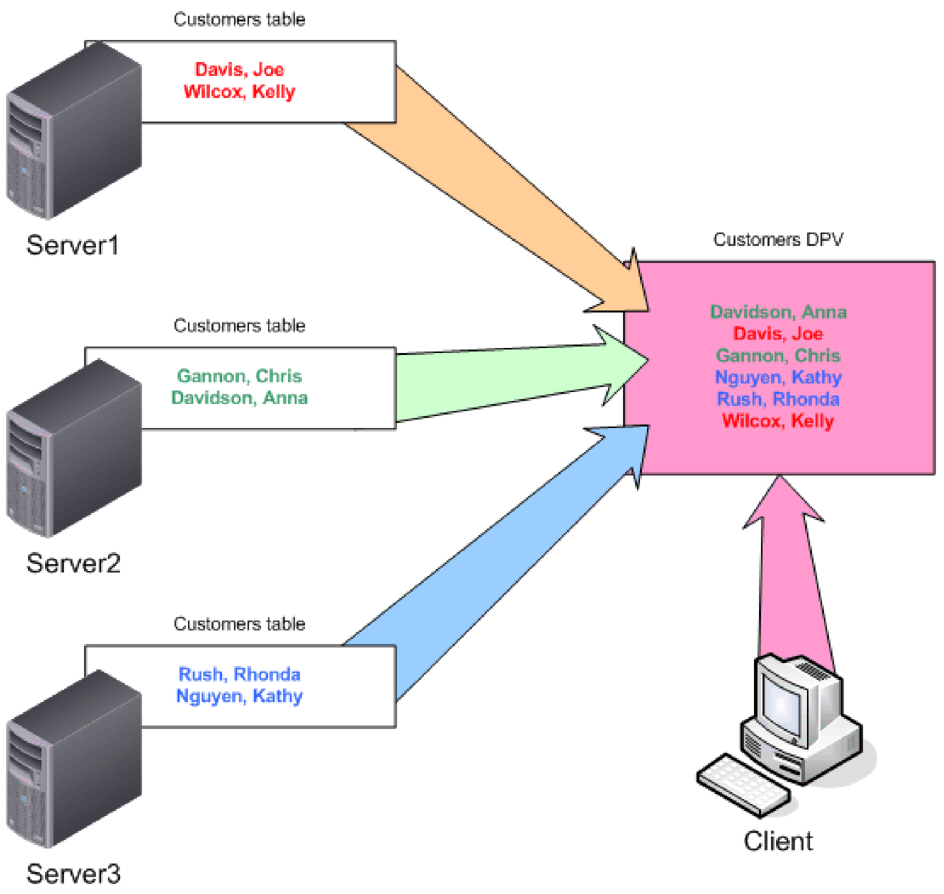

Figure 1.7: Distributed partitioned views enable a client to view three unrelated tables as one table.

Distributed partitioned views are a powerful tool for creating scaled-out applications. Each server participating in the distributed partitioned view is a *node*, and the entire group of servers is a *shared-nothing cluster*. Each server's copy of the distributed table (or tables) has the same schema as the other copies—the same columns, constraints, and so forth—but each server contains different rows.

It is crucial that tables are horizontally partitioned in such a way that each server handles approximately the same load. For example, an application that frequently adds new rows to the most recently added node places an unequal amount of INSERT traffic on that server, partially defeating the purpose of the scale-out strategy. If database reads primarily deal with newer rows, the new node will handle most of the traffic associated with the table, leaving the other nodes relatively idle. Appropriately redistributing the rows across the available nodes will alleviate the problem and more evenly distribute the traffic (and therefore the workload) across them.

Despite their utility, distributed partitioned views are only rarely used as a result of the difficulty inherent in evenly distributing rows across multiple servers. Failing to achieve an even distribution can result in disastrous performance, because certain participating servers will need to hold query results in memory while waiting for other servers that have more rows to process to catch up.

In Chapter 4, we'll explore distributed partitioned views—how to set them up and how to use them.

Distribution Partitioned Databases and Replication

Another scale-out approach involves partitioning a database across multiple servers, then replicating the database copies. Like the six-server 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.

Chapter 5 covers distributed partitioned databases in detail, including how-to information for creating and maintaining distributed partitioned databases.

Windows Clustering

Windows Clustering not only improves performance but can also be a useful technique for scaling out without increasing the risk of a server failure. For example, a two-node active/active cluster has two independent SQL Server machines. You can configure these nodes as a server farm, in which each server contains a complete copy of the database and users are distributed between them, or as 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 a catastrophe 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. That way, if a node fails, the other node can begin running at 100 percent capacity with about a 20 percent loss of efficiency. Even with this efficiency loss, performance is generally still well within an acceptable range, especially considering that applications after failover must run on half as much hardware.

Setting up clusters can be extremely complex. In the case of 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. To prevent confusion, it's advisable to buy an HCL-qualified cluster that is well documented from a major server vendor. This simplifies cluster setup and the vendor (and Microsoft) can provide cluster-specific technical support, if necessary.

Chapter 6 delves into Windows Clustering, along with a complete tutorial about how clustering works and how clustered SQL Server systems can be a part of your scale-out solution.

High-Performance Storage

High-performance storage offers an often-overlooked performance benefit for SQL Server—particularly external storage area networks (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, this type of storage is 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. Industry estimates suggest that a considerable amount of idle time in SQL Server results from waiting for the disk subsystem to deliver data. Improving the speed of the disk subsystem can, therefore, 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.

📖 Chapter 7 is all about high-performance storage and how it can help improve your scale-out solution.

Hurdles to Scaling Out Database Solutions

Although scaling out provides many benefits, the process can be difficult because so few databases are in a condition that lends itself to being scaled out. The following list highlights tips for scaling out database solutions:

- A database application that relies primarily on stored procedures offers greater flexibility than client software that sends ad-hoc queries to the database server. The reason is that much of the business logic is centralized on SQL Server and can more easily be changed to reflect a distributed environment.
- Indexes that are fine-tuned on a single server before distribution to multiple servers prevents poor index and database design decisions from becoming widespread problems.
- Databases that are overnormalized are possible—examine ways to reduce the number of joins. Ideally, average queries shouldn't join more than two or three tables.
- Split your application's tables across each file (or filegroup) appropriately. Indexes must exist with their related tables, and tables with foreign key relationships must be in the same file (or filegroup) as the lookup table. If you find that you can't divide the database, it will be a difficult task to distribute it.
- Views are an effective way to review data (and update data on SQL Server 2000), and SQL Server's distributed views capability allows clients to query the view without being aware of where the data physically resides.

An application that relies on views is well-suited for scaling out. For example, consider the example that Figure 1.8 illustrates in which the client computer submits a query for a particular view, which pulls data from three tables. The client sees these tables as a single virtual table and is able to work with, and even update, the data.

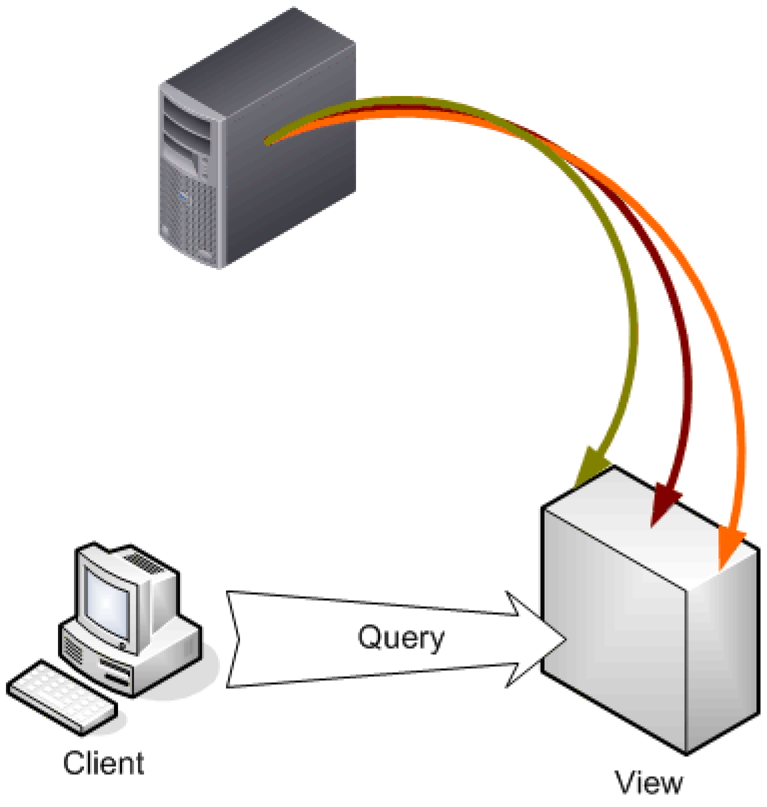

Figure 1.8: Client accessing data through a view.

Figure 1.9 illustrates the database distributed across three servers. The client, however, continues to access a view—a distributed view. The view pulls information from three tables on three servers, but no changes are necessary to the client.

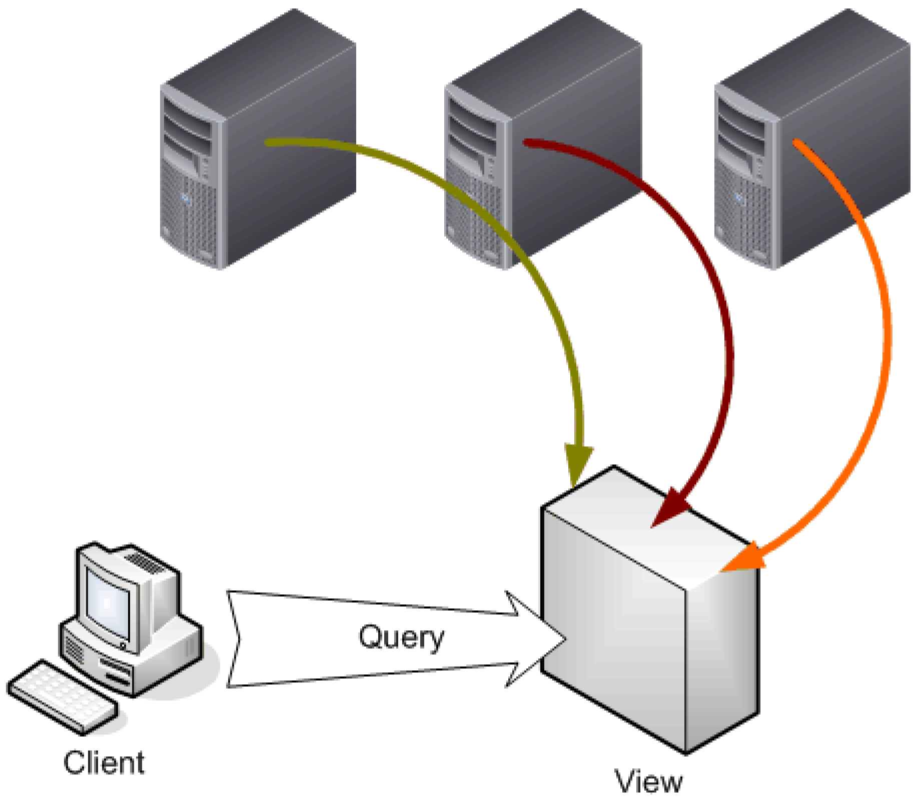

Figure 1.9: Client accessing distributed data through a view.

To clearly illustrate this point, let’s consider the alternative. Figure 1.10 shows a client directly accessing data from a server. This setup is inefficient, as ad-hoc queries require the most work for SQL Server to process.

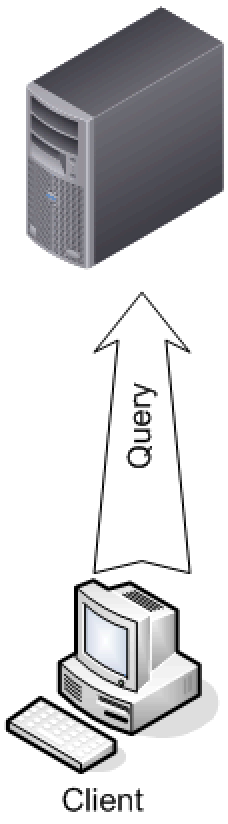

Figure 1.10: Client accessing data by using an ad-hoc query.

After the data has been distributed to three different servers (see Figure 1.11), the situation will degrade further. This situation would require a new or at least revised client application that is aware of the data distribution. In addition, the data will take longer to query, slowing application performance.

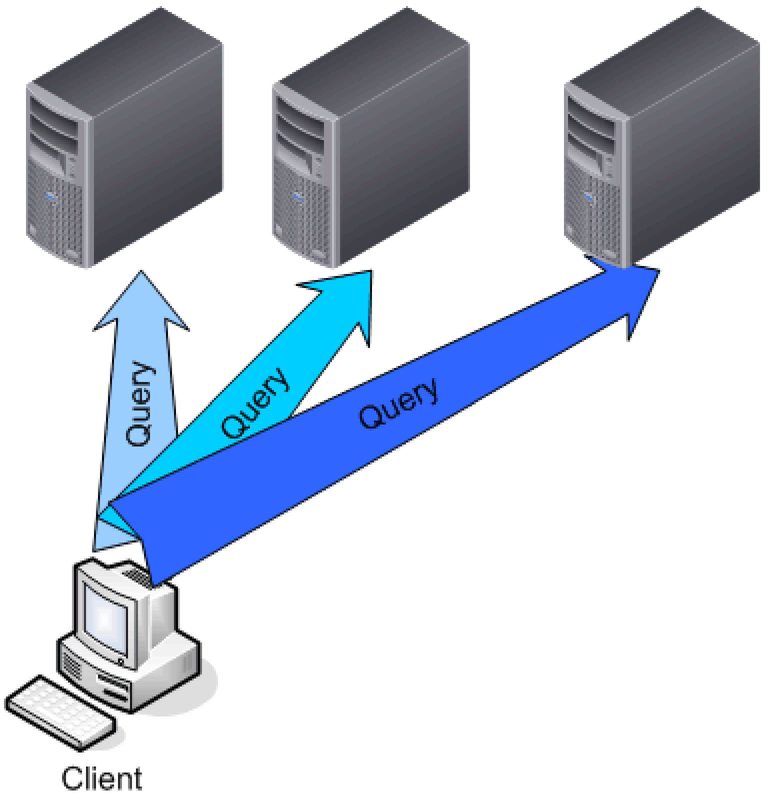

Figure 1.11: Client accessing distributed data by using an ad-hoc query.

To avoid the unpopular task of revising client applications, spend time analyzing the way your clients interact with a database. Use the results of this analysis to fine-tune the database before you consider a distributed data solution as part of a scale-out scenario.

> Chapter 3 will contain a more in-depth discussion of the challenges of scaling out SQL Server, including tips for getting your databases in shape for the move.

Of course, there's no step-by-step guide to scale-out success. Several hurdles exist that can make scaling out more difficult, either from a database standpoint or from a manageability standpoint. Solutions and workarounds exist for many of these hurdles, but it's best to put everything on the table up front so that you know what you're getting into.

Database Hurdles

Some databases are simply not built to scale-out easily. Perhaps the data can't be easily partitioned and divided across servers, or perhaps your circumstances make replication impractical. For many organizations, the database design and how it's used present the biggest hurdles to scaling out, forcing those organizations to do the best they can with scale-up solutions.

Even if a drastic redesign is required, however, SQL Server offers solutions that can make such a redesign possible and even relatively painless. Integration Services can be used to transform a problematic database design into one that's more amenable to scale-out, while views and stored procedures can help mask the database changes to client applications and middle-tier components, eliminating a cascade of design changes in complex multi-tier applications. Scale-out capability and performance starts with a solid database design, but SQL Server's toolset recognizes that few databases are initially built with scale-out in mind.

📖 I'll address initial database design issues in Chapter 3.

Manageability Hurdles

Manageability problems are often a concern with many scale-out solutions. Let's face it—scale-out solutions, by definition, involve adding more servers to your environment. You're then faced with the reality of managing multiple servers that have a complex set of relationships (such as data replication) with one another. These multiple servers require patch and software management, performance and health monitoring, change and configuration management, business continuity operations, and other complex tasks. SQL Server 2005 provides management capabilities that address some of these hurdles; proper management technique and third-party tools and utilities can help address other hurdles and make scale-out manageability easier and more comprehensible.

📖 Chapter 8 will tackle the complex topic of scale-out manageability, including its many difficulties and solutions.

Server Hardware: Specialized Solutions

After you overcome any database issues and have selected a scale-out technique, you can begin considering your hardware options. Can you create a scale-out solution using inexpensive PC-based servers? Scale-out solutions often lend themselves to less-expensive hardware more readily than scale-up solutions do, as scale-up solutions require you to squeeze every possible erg of performance from the server hardware.

Another option is specialized SQL Server machines. These systems are capable of running other applications, but have been fine-tuned to run SQL Server. A specialized SQL Server machine might provide the best overall performance, albeit at a much higher price.

Clusters present an additional hardware consideration. I recommend buying preconfigured clusters whenever possible, or at least servers with a certified cluster configuration and a detailed set of guidelines for configuring the servers in a cluster: they're easier to set up, easier to maintain, and easier to obtain support for if something goes wrong.

The ultimate server purchase (in terms of expense, at least) is Microsoft Datacenter Server, which can be scaled-out across high-end, proprietary hardware. Available in both Windows 2000 Datacenter Server and Windows Server 2003 Datacenter Edition, this hardware is sold in specially certified configurations with the OS preloaded. Generally, at minimum, these systems provide a large amount of data storage and are at least 4-way machines. This option is the most expensive and offers the most proprietary type of server. However, the hardware, OS, and drivers are certified as a package, so these systems are also the most stable Windows machines, earning 99.999 percent uptime ratings. Carefully examine your scale-out strategy to determine whether you need this much horsepower and this level of availability.

Microsoft's Datacenter program is not intended to equate with raw server power. Datacenter is intended primarily for reliability, by combining well-tested drivers, hardware, and OS components. However, due to the price, most Datacenter machines are also beefy, intended for heavy-duty enterprise applications.

I should also point out that Datacenter Server isn't specifically designed for scale-out solutions; servers running Datacenter are typically fully-loaded in terms of processor power, memory, and other features, and they are specifically designed for high availability. However, most scale-out solutions include a need for high-availability, which is where Windows Clustering can also play a role (as I'll discuss throughout this book).

Obviously, the money required for a Datacenter Server system—let alone several of them—is significant, and your solution would need extremely high requirements for availability and horsepower in order to justify the purchase. I suspect that most scale-out solutions can get all the power they need using PC-based hardware (also called *non-proprietary* or *commodity* hardware), running Standard or Enterprise Editions of Windows, using Windows Clustering to achieve a high level of availability.

For example, for the cost of a single data center server from one manufacturer, you can often configure a similarly equipped two-node cluster from other manufacturers. The clustering provides you with similar reliability for your application because it is less likely that both cluster nodes will fail at the same time as the result of a device driver issue. As the goal of scale-out solutions are to spread the workload across multiple servers, having more servers might be more beneficial than relying on one server. This decision depends upon your particular strategy and business needs. Scaling out allows for flexibility in the types of server hardware that you use, allowing you to find a solution that is specifically suited to your needs.

There is a whole new breed of server available to you now—64-bit. Although Intel's original Itanium 64-bit architecture remains a powerful choice, a more popular choice is the new x64-architecture (called AMD64 by AMD and EM64T by Intel). I'll cover this new choice in detail in Chapter 7.

Comparing Server Hardware and Scale-Out Solutions

Apples-to-apples product comparisons are difficult in the IT industry. Manufacturers boast of special bus architectures, memory caching techniques, and even specially modified hard drives. Manufacturers that develop *proprietary* hardware, meaning that the components are specially developed in-house, claim that these components provide a huge performance advantage. Regardless of whether these claims are true, proprietary hardware usually results in a higher price.

Another class of server vendors use *off-the-shelf* hardware. These vendors use chipsets, bus designs, and other components that are engineered and manufactured by companies such as Intel and AMD. The components are sold to a wider variety of manufacturers in much greater quantities, so the engineering costs of these components are lower than the costs of developing proprietary hardware, which, in turn, lowers the cost of buying a server built from off-the-shelf components.

Stability is an additional advantage to non-proprietary components. These components are in wide use, so the necessary software drivers are built-in to Windows in the form of Microsoft-signed drivers. In addition, bugs and other flaws are quickly found and corrected. Proprietary hardware has much less exposure and often requires special drivers from the manufacturer and specialized steps during the Windows installation.

Thus, comparing server hardware and scale-out solutions is a difficult task. Benchmarks, such as the TPC-C and TPC-H, might not be available for the precise machine configuration that you're considering (changing the amount of RAM technically invalidates the TPC benchmark results). Rather than attempting to adopt a solution without some type of comparison, you can use the following guidelines to help determine which solution is optimal for your situation.

Categorize the Choices

First, divide your vendor options—such as Dell, Fujitsu, Hewlett-Packard (HP), IBM, Micron, and Gateway—into proprietary and non-proprietary classifications. The non-proprietary hardware will be easier to compare, as the machines will use similar architectures.

Next, examine the prices for similarly configured machines. Your budget might eliminate high-priced contenders at this point. Also eliminate possibilities that don't meet any non-technical requirements (for example, you might require a specific level of support). With the remaining list, you can turn to benchmark comparisons.

Price/Performance Benchmarks

Start with TPC benchmarks that are appropriate to your application: TPC-C, TPC-H, or TPC-W. (We'll explore these benchmarks later in this chapter.) At this stage, focus on the price/performance ratio rather than raw performance. Use an Excel spreadsheet to create a chart like the one that Figure 1.12 shows as a useful tool for comparison.

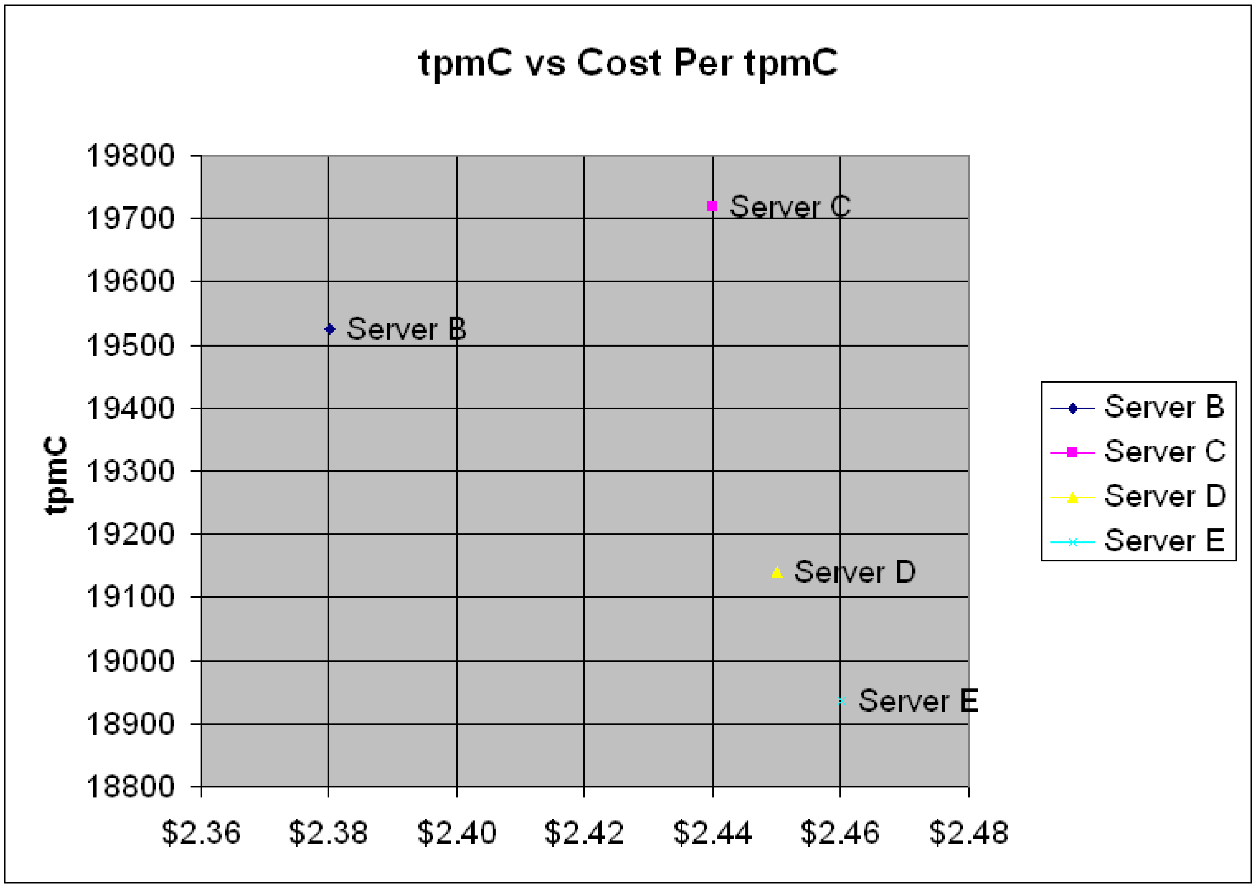

Figure 1.12: Create a price vs. performance chart.

In this chart, higher values on the Y axis mean better performance; dots further to the right cost more. The chart reveals that Servers B and C, which offer high performance at a lower cost, are good choices. Servers D and E fall into the high-cost category as well as provide low performance. This type of graph—called a *scatter graph*—can help you quickly compare the price/performance ratio of servers.

If TPC hasn't published results for the particular server in which you're interested, try grouping your option by manufacturer. Manufacturers typically have similar performance, especially within a product family. Take an average score for a manufacturer or product family as representative of the manufacturer or family's overall performance.

> Although these tests provide an "apples to apples" comparison of different platforms and manufacturers, it is important to note that the TPC configurations are not always applicable in real world environments. Therefore, using these test results for sizing is difficult. For sizing, it's best to ask the sales representative from one of the major vendors for tools—most major vendors have tools that will provide ballpark sizing information.

Identify the Scale-Out Solution

Next, focus on your scale-out solution. If you need to simply replicate data to multiple locations, determine whether you will need a different-powered server for each location, depending on the user population and workload you expect each server to support. To create a highly distributed database, examine the portion of the work each server is expected to perform to determine which hardware is appropriate. Your price/performance chart will help you identify good candidates.

In addition, if you're planning to implement database clusters, evaluate ready-made clusters that have been benchmarked. As Chapter 6 explores, building a cluster from scratch can be frustrating and result in poorer performance than a packaged solution.

Calculate a Total Solution Price

To calculate the cost of a potential solution, assemble several solution packages that include all the servers you'll need to implement your proposed scale-out solution. To include a performance consideration, give each solution package a performance score from 1 to 10 (best-performing servers receive a score of 10; the lowest-performing package is given a 1). This process provides an avenue of comparison for the packages based upon their total price, performance, and price/performance value.

> Manufacturers provide different support plans and maintenance agreements, making it a difficult task to compare servers' actual prices. I price solutions based entirely on the hardware, then add the cost of service or maintenance plans after a primary evaluation of the different solutions.

Take Advantage of Evaluation Periods

Once you've determined which package best meets your needs, negotiate payment terms. If you're making a large purchase, manufacturers may give you Net 30 (or longer) terms, with the right to return the products if you're not happy with them. Use this time to conduct a well-planned pilot of your new scale-out solution. Keep records of your pilot test as evidence for the manufacturer of exactly why you're returning the products.

If the purchase is very large and/or if you have a close relationship with a hardware vendor, it might be worthwhile to request a test lab. Microsoft has several labs in which the company offers racks and racks of storage and server equipment to customers who are developing database solutions. Taking advantage of this test lab is a great way to kick start development without investing up front in hardware—you can determine what you need before you buy it. Although Microsoft usually charges a fee for this access, it can be well worth the cost if the experience helps you avoid many of the common database design mistakes that are mentioned earlier in the chapter.

Industry Benchmarks Overview

You probably don’t have the time—or the money—to order a server from each major manufacturer in order to perform your own testing. Thus, you will need to rely on the results of industry benchmarks for your product comparison. The Transaction Processing Performance Council (TPC) is well-respected within the SQL Server hardware testing arena. TPC publishes benchmark results for several server platforms, providing an independent indication of the relative strengths of different servers. TPC has four benchmarks:

- TPC-C is the benchmark for basic transaction processing in any database application.
- TPC-H is the benchmark for decision support (data warehousing) databases.
- A related benchmark, TPC-R, analyzes performance for standardized report generation.
- TPC-W is the benchmark for Web-connected databases, particularly databases supporting an e-commerce application.

Although TPC benchmark scores are often used by vendors as a major marketing point, focus your attention on the benchmark scores achieved by the hardware you plan to employ in your scale-out solution. These benchmarks highlight the ability of hardware to run SQL Server efficiently; the more efficient, the higher the TPC score. TPC results that include large numbers are better, as they indicate an ability to process a greater number of transactions in a fixed period of time (1 minute). You should also note the price difference you pay for higher scores: the more a machine has been specifically designed for high SQL Server performance, the more you can expect to pay.

Earlier, this chapter noted that Web server farms are nearly all built with inexpensive hardware. The theory is that you can buy many inexpensive servers, and many servers are better than one.

Scaling out SQL Server means building a SQL Server farm, which doesn’t require you to pay extra for the most fine-tuned machine on the market. You can save money by buying machines that are based on standardized components (rather than the proprietary components used by most highly tuned machines).

TPC publishes results on its Web site at http://www.tpc.org. Results are broken down by benchmark, and within each benchmark, they are divided into clustered and non-clustered systems. However, TPC benchmarks are not available for every server, and TPC doesn’t test every server with SQL Server—the organization publishes results for other database software as well. TPC also publishes a price estimate that is based on manufacturer’s listed retail prices. This estimate enables you to quantify how much extra you’re paying for performance.

Separate benchmarks are available for many database applications. Although some of these applications can be based upon SQL Server, you’ll obtain more application-specific results by checking out the specific benchmarks.

Additional benchmarks exist, though few are geared specifically toward SQL Server. As you’re shopping for server hardware for a scale-out solution, ask manufacturers for their benchmark results. Most will happily provide them to you along with any other performance-related information that you might need.

Summary

In some cases, large database applications reach the point where scale up is not practical, and scaling out is the only logical solution. You might find, however, that scaling out is a more intelligent next step even when you could potentially squeeze more out of your existing design. The next chapter will explore tips and best practices for increasing efficiency on your existing servers, which is an important first step to creating a useful scaled-out database application.