

realtimepublishers.com<sup>tm</sup>

*The Definitive Guide<sup>tm</sup> To*

# Scaling Out SQL Server 2005

*Don Jones*

---

Chapter 8: High-Performance Storage.....	170
Storage Overview.....	170
Redundancy and Fault Tolerance.....	170
Performance .....	171
Storage and SQL Server.....	171
Comparing Storage Technologies.....	173
RAID.....	173
RAID 0.....	173
RAID 1 .....	174
RAID 4.....	175
RAID 5.....	176
RAID 10.....	178
Software RAID .....	180
Hardware RAID .....	181
SCSI Arrays .....	181
NAS.....	182
SANs.....	183
Specialized Storage.....	185
Design Principles .....	186
Best Practices .....	187
Summary .....	189

## Copyright Statement

© 2005 Realtimepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimepublishers.com, Inc. (the "Materials") and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimepublishers.com, Inc or its web site sponsors. In no event shall Realtimepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtimepublishers.com and the Realtimepublishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimepublishers.com, please contact us via e-mail at [info@realtimepublishers.com](mailto:info@realtimepublishers.com).

[**Editor's Note:** This eBook was downloaded from Content Central. To download other eBooks on this topic, please visit <http://www.realtimepublishers.com/contentcentral/>.]

## Chapter 8: High-Performance Storage

Applications such as SQL Server depend heavily upon the availability of high-performance storage. In fact, as much research and design has gone into creating high-performance storage subsystems as has gone into SQL Server, simply because most SQL Server applications eventually become storage-bound—meaning storage imposes the upper limit on application performance. Such is particularly true on 64-bit servers, where the greatly expanded memory capacity can help prevent SQL Server from becoming memory-bound.

This chapter will discuss the storage technologies that are most appropriate for use with SQL Server and introduce you to best practices for utilizing high-performance storage systems. These systems are critical to building successful scale-out solutions—without the proper storage subsystem backing up your SQL Server applications, all the advantages of replication, federated databases, and other scale-out technologies are degraded. Obviously, most of the storage suggestions in this chapter are also appropriate for scale-up solutions, as well.

### Storage Overview

Storage is, of course, the basis for most server applications. Ideally, we would be able to store all data in high-speed memory; but the cost of RAM is simply too high. Disks are much cheaper, albeit hundreds of times slower—RAM response times are measured in nanoseconds (ns) and disk response times are measured in milliseconds (ms). Disks are mechanical devices with moving parts, so they are also subject to more frequent failure than solid-state RAM, meaning loss of data is also a strong concern. Today's storage subsystems attempt to strike a balance between fault tolerance and performance.

### **Redundancy and Fault Tolerance**

The purpose of fault tolerance is to quite literally tolerate faults, or failures, in the system. For example, if a single drive fails, fault tolerance will prevent the data on that drive from being lost and prevent the server or application from crashing. Storage subsystems typically provide fault tolerance through the use of *parity* information or *mirrors*. Parity information is the result of a calculation performed on stored data and can be used to reconstruct portions of the stored data in a failure situation. Mirrors are online copies of data that are updated in real-time, providing a duplicate copy of the data that can be used if the original copy fails.

Fault tolerance involves copying some or all of the original data; thus, fault tolerance creates a negative performance impact. The resources used to write an extra copy or calculate and write parity information can't be used for reading or writing additional data.

## Performance

Performance is based on the idea of reading and writing data as quickly as possible to serve the users you need to support. As physical devices, storage subsystems have a number of elements that can impede performance:

- The disk, which spins at a fixed speed and cannot transfer data beyond that speed.
- The disk heads, which can be in only one place at a time and must waste milliseconds *seeking* the data that is desired. (Seeking is the process of moving the heads to the appropriate location of the disk so that the data spinning by underneath the heads can be magnetically read or modified.)
- Bus speed—the bus carries data between the disk and the controller hardware to which the disk is attached. Another bus connects the controller to the server, allowing Windows to communicate with the controller and move data to and from the disks.
- Device drivers are the actual software that communicates between Windows and the controller hardware. Poorly written device drivers can impede performance and are very difficult to pin down as a bottleneck.

Other elements—such as the fault tolerance scheme in use—can also impede performance by creating additional overhead within the storage subsystem. Optimal performance can be achieved in part by using better-performing components: faster disks, bigger busses, and less-obstructive fault-tolerance schemes. However, you can quickly reach the point at which you have acquired the fastest drives and widest data transfer bus and have implemented the fastest fault tolerance scheme available.

In such a case, better performance is possible through parallelism. For example, adding *more* disks to the system allows the server to save data to an idle drive when others are busy; adding controllers provides additional, parallel paths for data to come onto and off of disks. Many of the industry's top-performing storage solutions, in fact, rely heavily on sheer number of physical drives, utilizing multiple controllers per server, dozens of drives in cooperative arrays, and so on.

## Storage and SQL Server

Storage performance is absolutely critical to SQL Server's overall performance and to end user response times. For example, most administrators try to tune their queries to take advantage of indexes. Indexed queries are faster for SQL Server to execute but require many read operations from the storage subsystem.




Not all operations are disk-intensive; however, SQL Server is heavily dependent on good input/output performance. Thus, although not critical for every application, a good disk subsystem is an important asset.

First, SQL Server must read the root page of the index, then decide which page to read next. Index pages are read one at a time, creating a large number of storage operations, although each operation is relatively small—less than 10KB—in size. A typical index read in a relatively efficient index might require 100 read operations, which with a well-performing storage subsystem might take anywhere from half a second to a full second. In a slow subsystem, however, speeds can be as slow as a half second *per read operation*, meaning the same index search could take nearly a minute to execute. This scenario illustrates the level of performance difference a storage subsystem can have on SQL Server.

Less-efficient table scans can exacerbate the problems caused by a poor storage subsystem. Imagine a table scan that is going through a million rows of data—not uncommon in a large database—at the slow pace of 50ms per read. That could require SQL Server to spend an enormous amount of time—half an hour or more—to complete the table scan. You can test this sort of performance easily by installing SQL Server and a large database on a notebook computer. Notebooks typically have incredibly poor throughput on the hard drive—storage performance isn't really what notebooks are designed for. Fire off a query that table scans a million rows, and you'll see how bad things can get. Now image that a couple of thousand users are trying to do the same thing all at once, and you'll get an idea of how poor application performance would be as a result of an inefficient SQL Server storage system. As a result, database administrators and developers try to minimize table scans by providing SQL Server with appropriate indexes to use instead: Even on a speedy disk subsystem, table scans can rack up a lot of time.

So what constitutes good performance? Take a 15,000rpm disk drive, which is pretty standard equipment for high-end servers. The drive has a fixed performance level because it can only pull data off the drive as quickly as the drive's platters are spinning. A high-end drive might take 6ms to move the drive heads from one location on the platter to another, on average, which creates an approximate maximum throughput of about 160 read operations per second. The closer you get to this maximum, the poorer overall performance will be, so you should aim to stay within about 75 percent of the maximum, or about 120 operations per second.

Consider again an index read example with 100 operations; you can see that you're only going to get about one and a quarter index reads per second while staying within the safe range. This example illustrates how easy it is to reach the performance capacity for a single drive and why it's so important to use arrays of drives that work together rather than storing data on a single drive.

 SQL Server 2005 introduces built-in partitioning, which allows you to more easily spread a database across several disks while still managing the database as a single set of objects. However, in large databases simply spreading the database across two or more single disks still won't provide a significant performance increase. Instead, you'll find yourself spreading the database across multiple disk *arrays*, or, more commonly, using large arrays (such as in Storage Area Networks—SANs, which I'll discuss in a moment) and simply ignoring SQL Server's partitioning capabilities.

## Comparing Storage Technologies

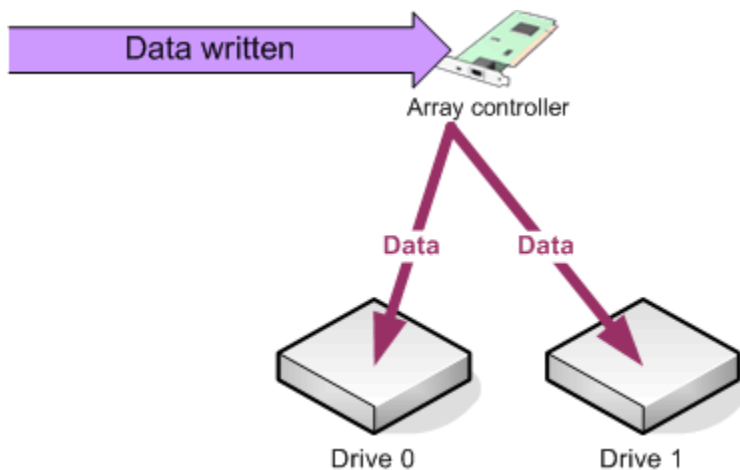
A number of technologies have been created to improve storage performance. The most fundamental of these is a Redundant Array of Inexpensive Disks (RAID). Other technologies, such as Storage Area Networks (SANs), improve the manageability and performance of storage. The following sections discuss each of these technologies, explain how they work, and explain their role in a SQL Server scale-out environment.

### RAID

RAID is the cornerstone of most high-performance storage solutions. The idea behind RAID is to utilize multiple disks in concert to improve both redundancy and performance. RAID defines several different levels, which each provide a tradeoff between redundancy and performance. In a production environment, you're likely to encounter RAID 1, RAID 5, and RAID 10.

### RAID 0

RAID 0 uses a technique called *disk striping*, which Figure 8.1 illustrates.



**Figure 8.1:** RAID 0.

As data is streamed to the controller, it is divided more or less evenly between all the drives in the array (two are required). The idea is to get more drives involved in handling data to increase overall throughput. When data is sent to Drive 0, the controller would normally have to wait until Drive 0 accepted that data and finished writing it. In RAID 0, the controller can move on to Drive 1 for the next chunk of data. RAID 0 improves both read and write speeds but has an important tradeoff: if a single drive fails, the entire array is pretty much useless because no one drive contains any entire file or folder. Thus, although RAID 0 improves performance, it doesn't improve redundancy. This concept of disk striping is an important one that comes up again in other RAID levels.

The performance of RAID 0 improves as you add drives. In a 2-drive system, for example, the controller might submit data to both Drive 0 and Drive 1 and still need to wait a few milliseconds for Drive 0 to catch up and be ready for more data. With a 4-drive array, Drive 0 is much more likely to be waiting on the controller, instead. With 8 drives, the odds improve even more, practically ensuring that Drive 0 will be ready and waiting when the controller gets back to it. It is possible to reach an upper limit: many lower-end controllers, for example, reach their own maximum throughput with 7 to 10 drives attached, meaning you're not going to see a performance improvement by attaching any drives beyond that point.

## RAID 1

RAID 1 is also called *mirroring*, and is illustrated in Figure 8.2.

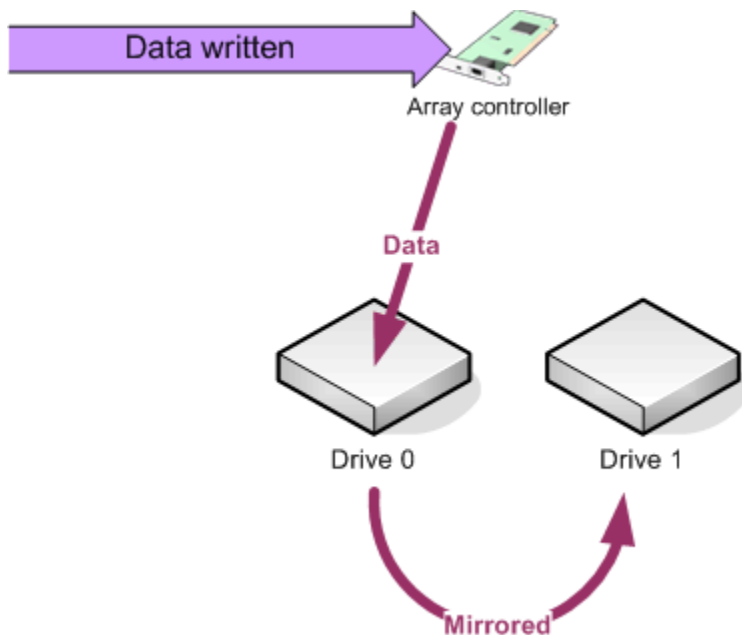


Figure 8.2: RAID 1.

In this level of RAID, the controller writes data to and reads data from a single disk. All written data is also written—or *mirrored*—to a second disk. Should the first disk fail, the second is available essentially as an online, up-to-date backup. Most array controllers will allow the server to function normally off of the mirror until the failed disk is replaced.

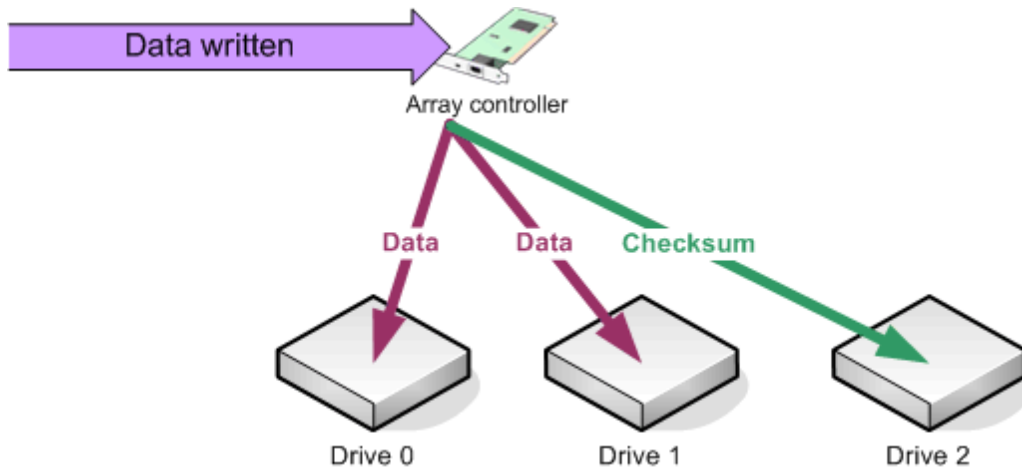
RAID 1 is almost the opposite of RAID 0, in that it provides *no* performance advantage, but does provide redundancy; the failure of a single disk won't harm the server's operations. RAID 1 can, in fact, *reduce* performance slightly in a write-heavy application such as SQL Server because all data is written twice. Most high-end RAID controllers, however, can minimize this performance impact through creative management of the data bus used to communicate between the controller and the disks.

Increasing the drives in a RAID 1 array does improve fault tolerance. For example, 3 drives in an array would survive the failure of any 2 drives. That's pretty expensive insurance, however, and not at all common in production environments. In addition, more drives in a RAID 1 array can reduce performance due to the additional write operations required.




## RAID 4

RAID 4 takes RAID 0 one step further by adding *parity* to disk striping. Figure 8.3 shows how RAID 4 works.



**Figure 8.3: RAID 4.**

RAID 4 uses the same disk striping mechanism that RAID 0 uses. However, RAID 4 also utilizes an additional disk to store *parity* information. If a single data disk fails, the parity information, combined with the data on the remaining disks, can be used to reconstruct the data from the missing disk. This technique allows the array to continue operating if a single disk fails. However, because a single disk is used to store all parity information, incoming write operations can't be easily interleaved. In other words, the parity disk creates a bottleneck that, in write-intensive applications, can partially or totally defeat the performance improvement offered by disk striping.

 RAID 4 requires a minimum of 3 drives: 2 for striping and 1 for parity. Like most other RAID levels, all drives in the array must be of equal size.

### How Parity Works

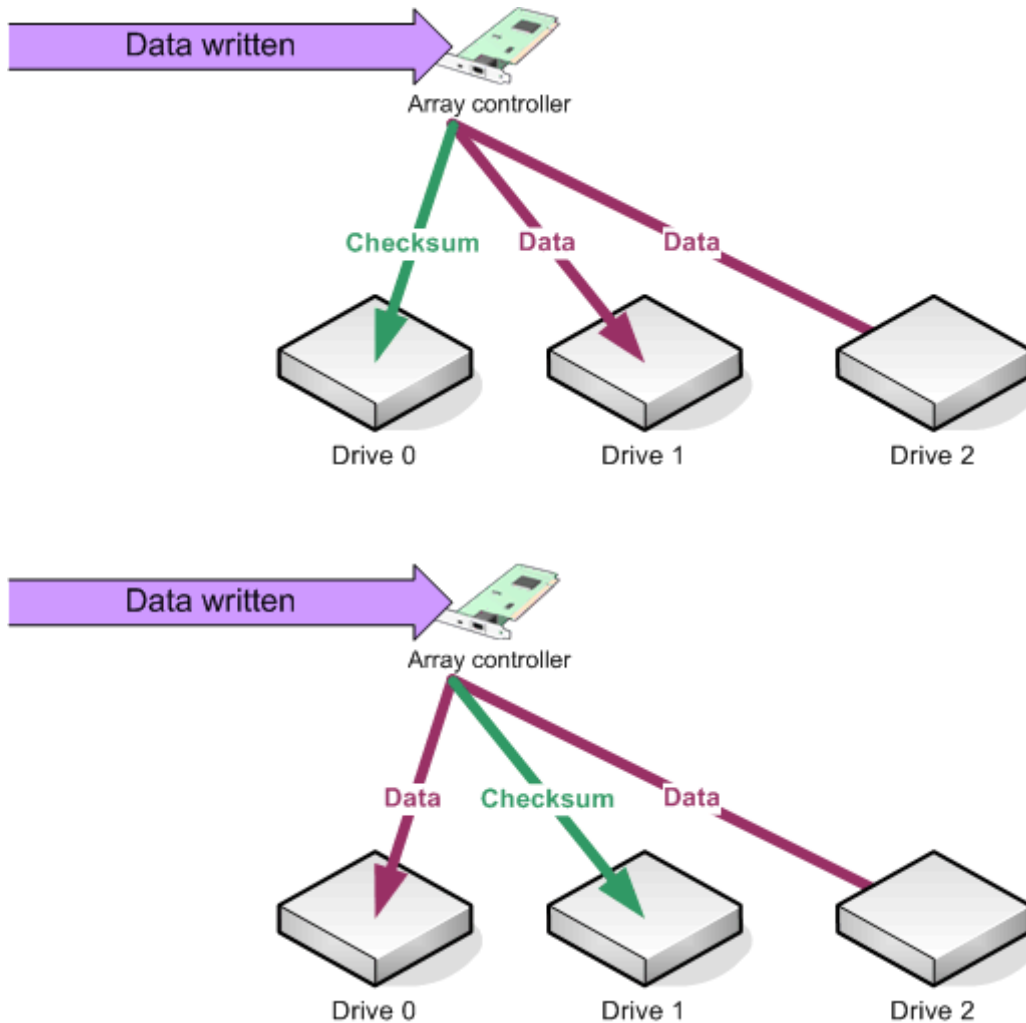
Parity works by performing mathematical calculations on a *stripe*. Suppose that you have a RAID 4 array with 4 data drives and a 5 drive for parity storage. Now suppose that you write a couple hundred kilobytes of data to the array. The first block of data will be written to drive 0, the next to drive 1, and so forth. The first four blocks of data, which are written to drives 0 through 4, are referred to as a stripe. The controller then performs a mathematical calculation on that stripe to arrive at a checksum, or parity block.

For example, suppose that the data written to drive 0 was the number 1. Drive 1 got the number 2, drive 2 got 3, and drive 3 received the number 5. Adding the values in the stripe yields the number 11 ( $1+2+3+5$ ). That result—the checksum, or parity—is stored on drive 4.

If Drive 2 fails, a piece of every stripe is now missing. The controller can, however, re-create that information by reversing the parity calculation. Substituting  $x$  for the data in the missing stripe, the calculation is  $1+2+x+5=11$ . Basic algebra tells us that  $x$  must equal 3, which is the data from the missing portion of the stripe. If *two* drives fail, the parity information is no longer sufficient to reconstruct the missing data; the equation might be  $1+2+x+y=11$ , leaving more than one possible value for both  $x$  and  $y$ . Although this example is an oversimplification of the actual math used to create parity information, it is an accurate illustration of how this relatively simple technology can be used to re-create missing data.


## RAID 5

RAID 5 offers a better performance-and-redundancy compromise than RAID 4 offers. Rather than using a single drive for parity information, RAID 5 rotates the parity information across the drives in the array, essentially striping the parity information along with the actual data being written. So for the first chunk of data sent to the controller, the disk operations look like those that Figure 8.3 shows. Figure 8.4 shows the subsequent write operations rotate the drive containing the parity information.



**Figure 8.4: RAID 5.**

The net effect of RAID 5 is that a single drive can fail and the array can remain functional by using the parity information to reconstruct the missing data. RAID 5 can handle interleaved write operations, improving its performance over RAID 4. However, because the parity information still represents an extra write operation, write performance is still slightly slower than read performance in a RAID 5 array. In practice, RAID 5 offers perhaps the best tradeoff between performance, cost, and redundancy for most server operations.

 Although RAID 5 requires a minimum of 3 drives to operate, it works better with more drives. Common RAID 5 arrays will have 7 or 8 drives in total, and can have many more, depending on the capabilities of your hardware.

## RAID 10

RAID 10 is a combination of RAID 1 and RAID 0—hence the name. Figure 8.5 shows how RAID 10 works.

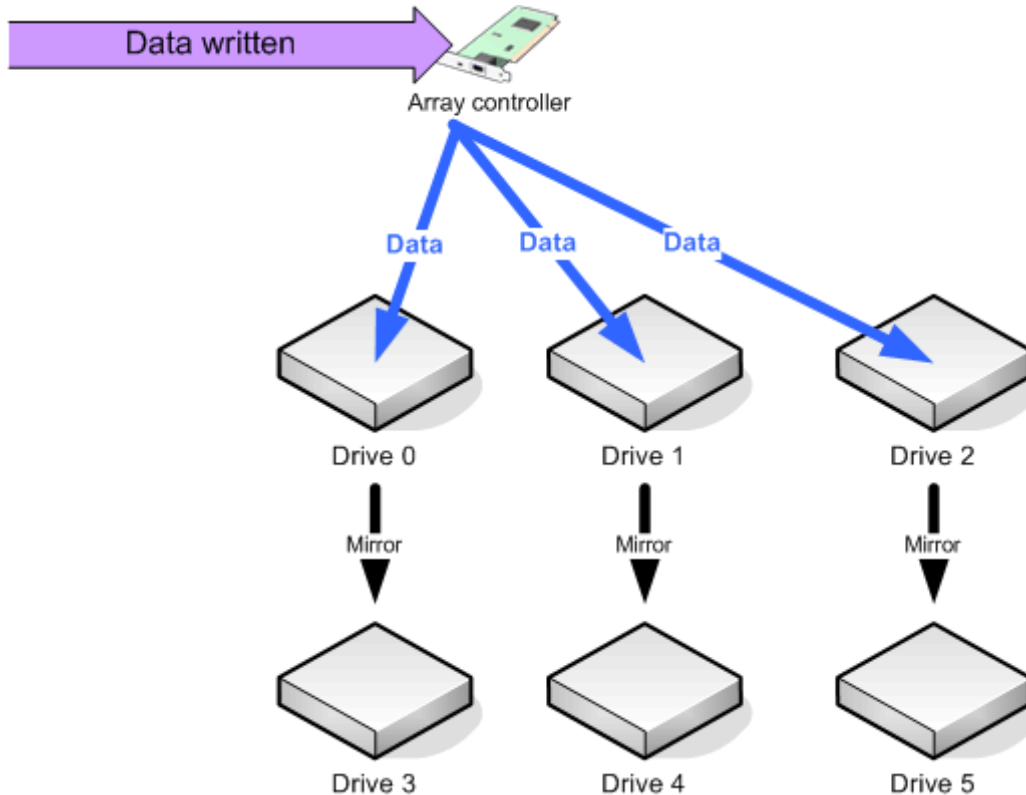


Figure 8.5: RAID 10.

A RAID 10 array is essentially two parallel arrays. The first array is a RAID 0 array, which uses disk striping—without parity—for maximum performance. The second array is a mirror of the first, providing the high level of fault tolerance offered by RAID 1. Parity information isn't required because each drive in the RAID 0 array has its own dedicated mirror; this type of array can theoretically survive the failure of every single drive in the main RAID 0 array, because each is backed up by a dedicated mirror drive.

👉 RAID 10 provides underlying technologies for many high-performance backup systems that are specific to SQL Server. For example, although SQL Server provides the ability to back up databases while they are in use, doing so reduces server performance. An integrated storage and backup solution can implement a third array as a second mirror set in a RAID 10 configuration. When a backup is required, the third array is detached (or the *mirror is broken*) and becomes a static snapshot of the data in the array. This third array can be backed up at leisure, then reattached (the *mirror repaired*) when the backup is complete.

RAID 10 provides the best possible performance and redundancy for SQL Server applications, but it does at a hefty price: You must buy twice as many drives as you need to store your databases. Still, in a high-end database, the price is usually well worth the benefit.



Various levels of RAID offer price tradeoffs. In RAID 1, for example, you "lose" the space of an entire drive to the fault-tolerance scheme. With RAID 4, the parity drive is "lost" to you, because it is dedicated to parity data. RAID 5 spreads out parity information, but you still "lose" the space equal to one entire drive. With RAID 10, you "lose" half of the drive space you bought—in exchange for better performance and fault tolerance.

Performance can vary significantly between different RAID 10 implementations. For example, some implementations use a single controller that issues write commands individually to each drive in the arrays. So for each write operation, two write commands are issued: one to the main RAID 0 array and another to each drive's mirror. More advanced implementations provide better performance by eliminating the extra write command. In this implementation, the two arrays function independently; the second array responds only to write requests and simply watches for commands sent to the first array, then carries out those commands in parallel. Because most SQL Server applications tend to be write-heavy, this latter, more advanced type of array is preferred.



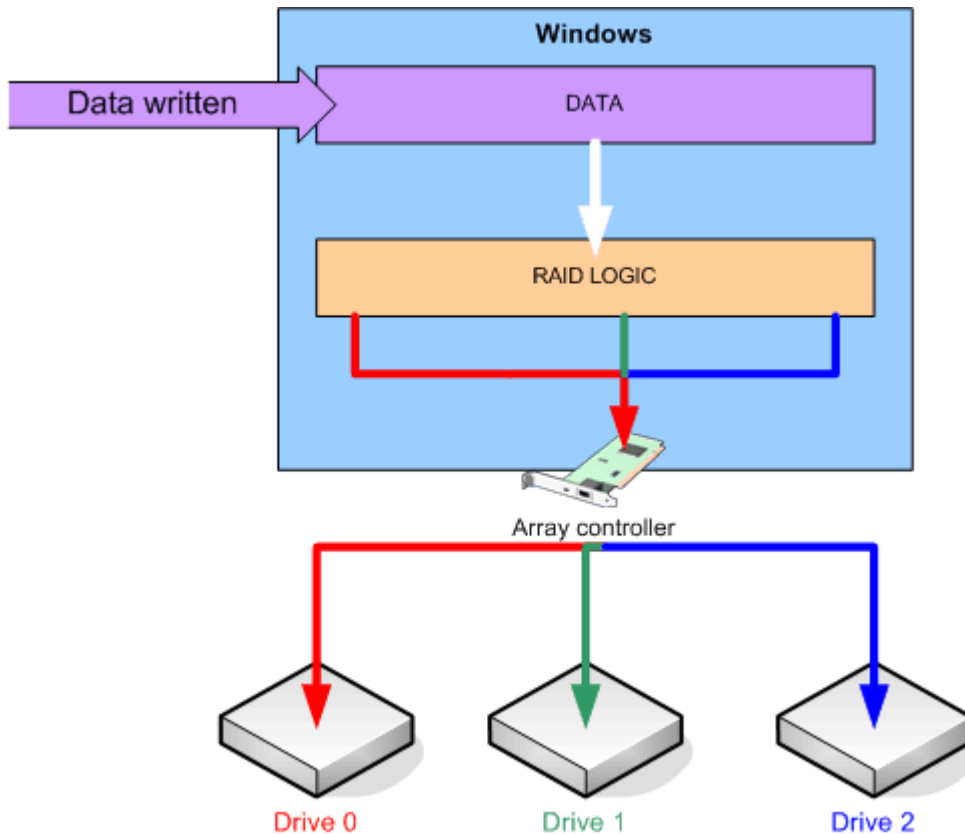
It's actually uncommon to see RAID 5 in use in actual, production high-end databases. RAID 10 is a much more common solution, as it provides both excellent performance and fault tolerance.

Another technique is to use dual RAID controller cards. Windows issues write commands, which the cards' device driver accepts and passes on to the controllers in parallel. The controllers then direct their attached drives independently, improving performance because write commands don't need to be duplicated. In some implementations, read commands are also carried out in parallel; the driver accepts read data from whichever array responds first, ensuring that the failure of even an entire array creates no additional lag time for the application. The device drivers can also implement a level of error correction by comparing the read results from both arrays to ensure that they're identical. This error-correcting can create a performance hit for read operations, because the controllers' driver must wait for both arrays to respond before delivering data to Windows.

The lesson to be learned—particularly from RAID 10, which offers the broadest number of implementation variations—is that you need to study and understand how various implementations work, then select the one that best fits your business needs. When combined with options such as SANs (which I'll cover later in this chapter), the possible implementations can become perplexing, so don't hesitate to ask vendors and manufacturers to explain exactly how their solutions work.

## Software RAID

Windows includes its own software-based RAID capabilities. These capabilities allow you to use standard IDE or SCSI drives attached on a non-RAID controller card and still have the fault tolerance benefits of a RAID 1 or RAID 5 array or the drive flexibility of a RAID 0 array. Figure 8.6 illustrates how Windows logically implements software RAID.



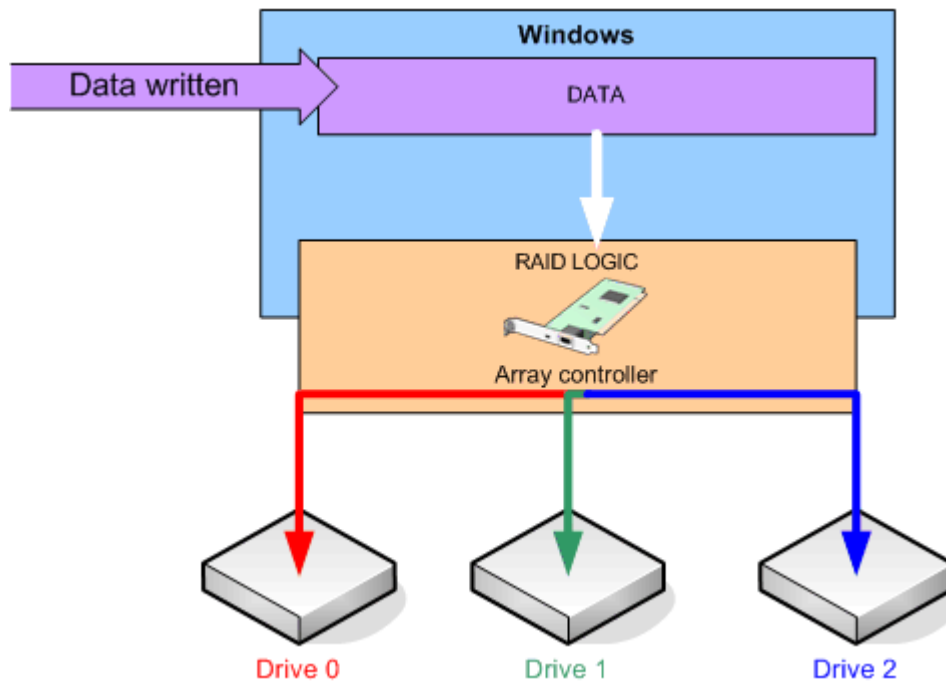
**Figure 8.6:** Windows software RAID logical flow.

As this figure illustrates, Windows implements the RAID logic in software. As data is written, Windows decides to which drives the data will be written, and sends each data stream to the controller card. The controller writes the data to the specified disk.

Because Windows itself is performing the RAID logic, it's fairly inefficient. Windows is a software application running on the server's processors; the RAID logic must pass through several instruction sets before finally having an effect on the underlying server hardware. In terms of performance, Windows RAID is about the worst single thing you can do to any SQL Server computer, particularly one participating in a scale-out solution that is supposed to offer improved performance.

## Hardware RAID

In a hardware RAID solution, the RAID logic is moved from Windows to a dedicated processor on a RAID-capable controller card. This card presents the array to Windows as a single physical disk and handles the task of splitting the incoming data stream across the drives in the array. Because the RAID logic is executed directly in hardware—and dedicated hardware, at that—its performance is much, much faster than software RAID. Figure 8.7 illustrated the logical flow of data.



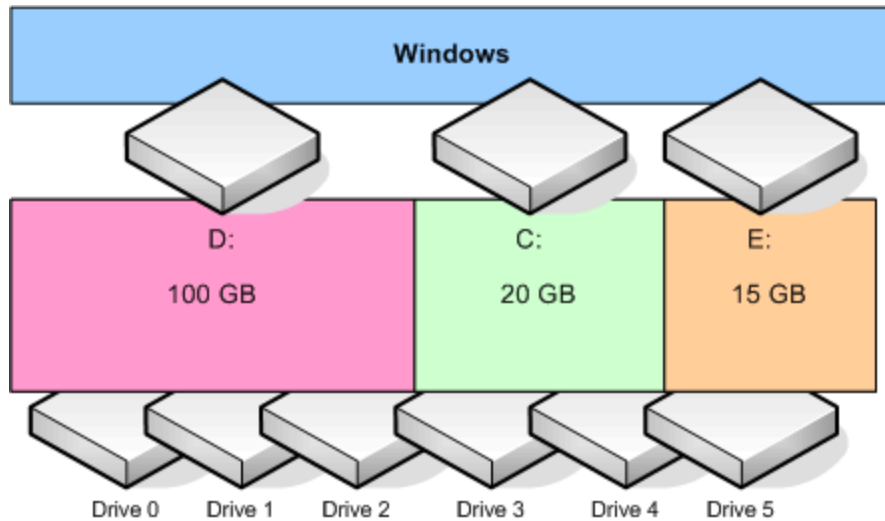
**Figure 8.7: Hardware RAID logical flow.**

Notice that, in this case, the array controller is a card installed directly in the server. This scenario is common for a traditional SCSI array; Network Attached Storage (NAS) and SANs—which I’ll discuss momentarily—work a bit differently than a traditional SCSI array.

## SCSI Arrays

SCSI arrays are perhaps the most common type of arrays in most data centers. These can take the physical form of an external array box, filled with drives and connected by a copper-based SCSI cable. Many servers also offer internal SCSI arrays that can hold anywhere from 2 to 12 drives.

SCSI arrays hold all the basic concepts for arrays. For example, Figure 8.8 shows how arrays are created from physical disks—in this case, 6 of them—then logically partitioned by the array controller. Each partition is presented to Windows as a single physical device, which Windows can then format and assign a drive letter to.

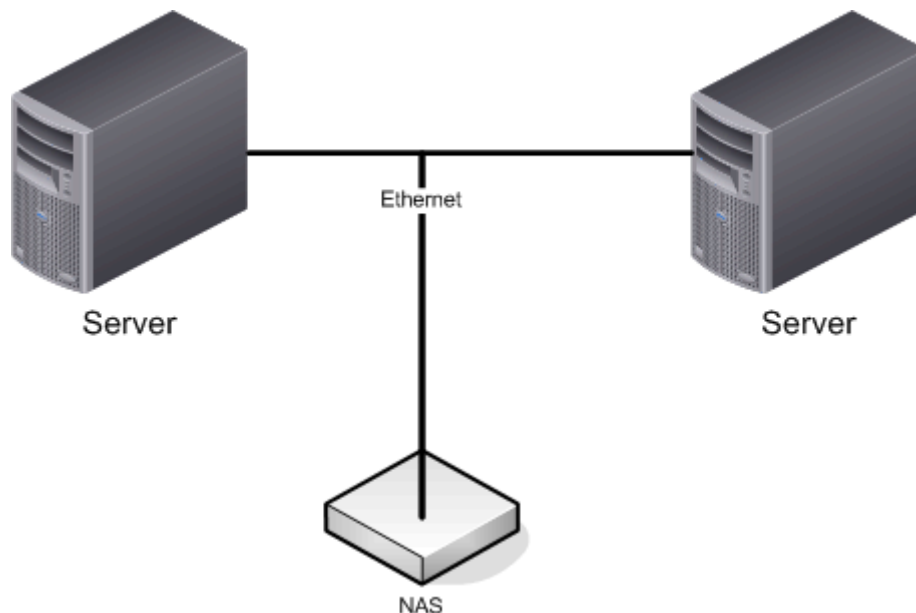


**Figure 8.8: Differences between physical and logical drives in arrays.**

In this example, the 6 physical drives might be configured as a single RAID 5 array, which is then divided into logical partitions to provide different areas of space for an application. One restriction common to most SCSI arrays is that the array must be physically attached to the SCSI controller, and only one computer—the one containing the controller card—can utilize the array. Such is not necessarily the case with NAS and SANs.

## NAS

NAS consists of a dedicated storage device that attaches directly to a network. Generally, these devices contain some kind of embedded server OS, such as Linux or Windows Embedded, which enable them to act as a file server. Because they function just like a file server, they are accessible by multiple users. In theory, SQL Server can store databases on NAS (see Figure 8.9).



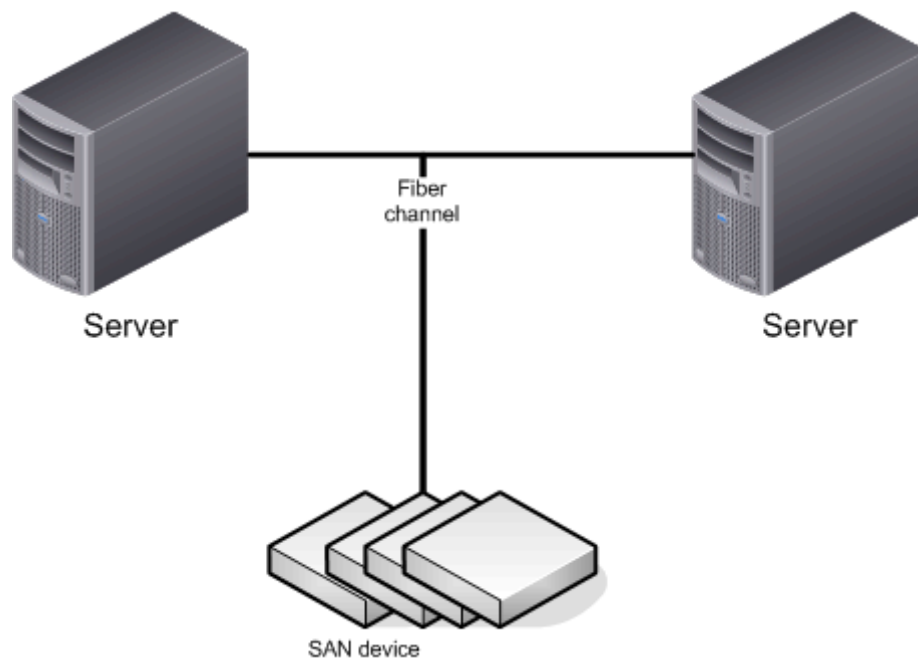
**Figure 8.9: An example NAS configuration.**



However, NAS has some major drawbacks for use in any SQL Server solution, particularly a scale-out solution. First, NAS devices are usually designed and built to replace file servers; they might offer a level of redundancy by incorporating RAID, but they aren't meant for blazing disk performance. Another drawback is the fact that data must reach the NAS device by passing over an Ethernet network, which is perhaps one of the least efficient ways to move mass amounts of data. Thus, although a NAS device might make a convenient place to store oft-used SQL Server scripts or database backups, it is a very bad location to store SQL Server databases.

## SANs

SANs appear superficially to be very much like NAS, as Figure 8.10 shows. However, there is a world of difference between SANs and NAS.

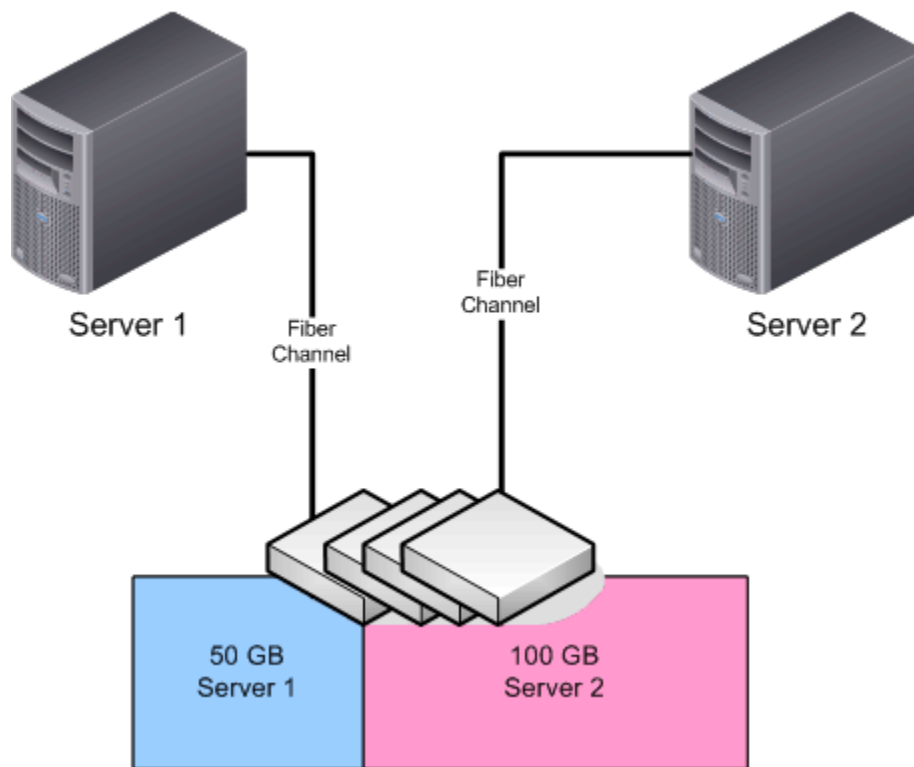


**Figure 8.10:** An example SAN configuration.

In a SAN, disks reside in dedicated array chassis, which contain their own internal controller boards. These boards are often manageable through an external user interface, such as an embedded Web server or a configuration utility. The boards control the primary operations of the array, including its RAID level and how the available space is partitioned. The array is connected—often via fiber-optic cabling in a typical fiber channel SAN—to a hub. Servers contain a fiber channel controller card and are connected to the hub. In effect, the SAN is a sort of dedicated, specialized network that in many ways resembles the infrastructure of an Ethernet LAN.

Windows reads and writes data by sending normal storage requests to the fiber channel controller card driver. The driver communicates with the controller hardware, and places storage requests onto the fiber channel network. The other devices on the network pick up these requests—just like a client computer might pick up network traffic—and respond appropriately. Even at this logical level of operation, SANs don't appear to be incredibly different from NAS devices; the main difference is in speed. Fiber channel networks can carry data *hundreds* of times faster than common Ethernet networks can, and fiber channel networks are optimized for dealing with high volumes of data, such as that from a SQL Server application. SAN implementations also tend to come with exceedingly large memory-based caches—in the gigabyte ranges—allowing them to accept data quickly, then pool it out to disks.

SAN arrays can be shared between multiple servers. As Figure 8.11 shows, the array's controller board determines which server will have access to which portion of the available data. In effect, the SAN array partitions itself and makes various partitions available to the designated servers on the fiber channel network.



**Figure 8.11: Partitioning space in a SAN.**

This ability for a SAN to be “shared” (the space isn't really shared, but partitioned between servers using the SAN) makes SANs a very effective tool for reducing management overhead: The entire SAN can be managed (and often backed up, depending on the solution) as a whole, and can often be re-partitioned somewhat dynamically, providing you with the ability to provision more space for servers that need it.

☞ SANs require software in order to run; look for SAN controllers that embed as much of their operating software as possible in their controllers or in the array hardware. This setup avoids placing any server-based software—and thus, overhead—onto your SQL Server computers, conserving as much processing power as possible for SQL Server itself.

Also, look for SAN solutions that offer large caches. A cache—an area of fast memory that accepts and holds data until the relatively slower hard drives can accept it—allows SQL Server to write data quickly, faster even than the storage solution's drives can really accept it. Simple RAID controllers typically offer caches, too, but often in the megabyte range. A high-end SAN solution can offer tremendously larger caches, increasing the amount of data that can be cached and improving SQL Server's performance considerably.

Interestingly, the advent of high-speed Gigabit Ethernet (GbE) technologies is making Ethernet a contender for creating SANs. Ethernet offers significantly lower pricing than many fiber channel-based SANs because Ethernet is a decades-old technology that runs over much less expensive copper cabling (fiber channel can also be implemented over copper but, for performance reasons, almost never is). A technology called Internet SCSI (iSCSI), which is supported in Windows Server 2003 (WS2K3—with the appropriate drivers), allows a server to address networked SCSI-based arrays over an Ethernet connection. Generally, this Ethernet network would be dedicated to SAN purposes, and servers would have an additional network adapter for communicating with clients.

Other emerging technologies include network adapters that have the ability to move data from a networked array directly into the server's memory. Referred to as *TCP/IP offloading*, this technology promises significant performance improvements for storage operations (as well as other data-intensive applications) because the server's processor can be bypassed completely, leaving it free to work on other tasks while data is moved around. Keep an eye on these emerging technologies as they mature to see what the next generation of high-performance storage will offer.

### **Specialized Storage**

You can now purchase preconfigured specialty storage solutions. These are generally packaged systems that combine basic RAID technology with proprietary controllers to improve performance, provide better redundancy, and so forth. For example, EMC offers several SAN solutions. These solutions include high-end manageability, meaning they can be incorporated into enterprise management frameworks such as Hewlett-Packard OpenView, or by monitoring and health solutions such as Microsoft Operations Manager (MOM), or monitoring solutions from companies such as Altiris. They can also include high-end fault protection, such as the ability to send an alert to a service technician when a drive fails. In fact, the first you hear about a failed drive might be when a technician shows up that afternoon with a replacement.

These high-end storage systems can, in fact, provide additional scale-out capabilities, redundancy, and performance above and beyond simple RAID levels. EMC's CLARiiON CX systems include proprietary software that provides full and incremental replication capabilities—even across long-distance WAN links—allowing you to maintain a mirror of your data at a remote location for additional fault tolerance and business continuity.



Dell offers branded versions of several EMC solutions, such as the Dell/EMC AX100, which comes in both Fibre Channel and iSCSI versions, and the Dell/EMC CX700. Other vendors offer proprietary solutions with similar capabilities.

## Design Principles

To ensure an optimal high-performance storage design for your environment, consider a few basic design principles. By reviewing the basic storage technologies, you can see that there are several potential performance bottlenecks that any design should strive to work around:

- **Controllers**—Controllers represent a single point of contact between a storage subsystem and Windows (and, therefore, SQL Server). In other words, all data has to pass through a controller; if you have only one controller, it will represent a potential bottleneck for data throughput.
- **Drives**—Drives have a fixed performance maximum, which simply cannot be overcome. Controllers can help alleviate this bottleneck by caching data in the controller's onboard RAM. In addition, you can implement arrays of disks to remove the bottleneck represented by a single disk.
- **Bandwidth**—The path that carries data from the controller to the drives is another potential bottleneck. IDE, the most common drive technology in client computers, is extremely slow compared with other technologies and isn't suited for use in servers. Newer versions of SCSI are faster, and some SAN technologies are faster still.

Your high-performance storage design should also consider the various types of data that SQL Server deals with:

- **Databases**—Tend to be read-heavy in online analytical processing (OLAP) applications, or mixed read- and write-heavy in online transaction processing (OLTP) applications
- **Transaction logs**—Write-heavy in OLTP applications
- **OS files**—For example, the page file, which is read- and write-heavy, especially in systems with insufficient RAM

The ideal SQL Server storage subsystem, from a performance and fault tolerance point of view, might include the following:


- A RAID 1 array for the OS files, including the page file—You could increase performance by placing the page file on a RAID 0 array, but you'll lose fault tolerance. If a drive in the RAID 0 array fails, you probably won't lose much data but the server will go offline. Use a dedicated controller—servers will often have one built-in—for this array.
- A RAID 5 array for the transaction log—Although RAID 5 imposes performance penalties for writes (as do many RAID levels), it provides a good balance between fault tolerance—essential for transaction logs—and write performance. If money is no object, create a small RAID 10 array for transaction logs and you'll get better performance and fault tolerance. Regardless of which you choose, use a dedicated controller for this array.
- Multiple RAID 10 arrays for databases, or RAID 5 if RAID 10 is too expensive—The goal is to figure out how much space you need, then use a larger number of smaller drives to improve overall throughput. Ideally, each array should be on a dedicated controller to provide independent bandwidth. For large databases, try to split the database's tables into multiple secondary database files and spread them across the available arrays so that the workload tends to be evenly distributed across the arrays. Ideally, these RAID 10 arrays can be implemented in fiber channel SANs, providing better throughput than copper-based SCSI connections.

## Best Practices

Over the years, best practices have been developed with regard to storage performance in SQL Server. These practices mainly come from long experimentation and reflect a sort of combined experience from within the SQL Server community. By following these practices, you can help ensure the best possible storage performance from your SQL Server scale-out solution.

- Do not under any circumstances use Windows' built-in software RAID capabilities. They're simply too slow. Use only hardware RAID controllers that implement the RAID array and present Windows with what appears to be a single physical disk that represents the entire array. All major server manufacturers offer these controllers in their servers. Or, simply use a SAN, which internalizes all the RAID capabilities and presents a single logical drive to Windows.
- Use the fastest disks possible. 5400rpm and 7200rpm disks are commonly available, although less than suitable for server use, where 10,000rpm is generally the accepted minimum. SCSI disks exceeding 15,000rpm are common in server configurations. The ability of the disk to spin its platters faster also gives it the ability to transfer data on and off those platters faster. Provided the drive is connected to a fast controller capable of handling the drive's throughput, faster drives will always result in increased performance.
- Don't mix drives in an array. Whenever possible, every drive in an array should be the same size, speed, and brand, ensuring the most consistent possible performance across the array. Mixing drives will lead to inconsistent performance, which can create performance hits that are difficult, if not impossible, to positively identify.

- The more disks, the merrier. Computers can generally stream data to and from disks faster than even the fastest disk can handle; by implementing arrays with more disks, the computer will be able to move on to the next device in the array while the last device “catches up.” Arrays of at least seven to eight disks are recommended, and larger arrays are common in specialized storage solutions such as those sold by EMC.
- Disk controllers are a bigger bottleneck than many administrators realize. Select a controller that has been tested and demonstrated high performance numbers. The controller should have its own CPU, and, ideally, should be bus-mastering, providing it with dedicated access to the server’s memory and offloading work from the server’s CPUs. As I’ve already mentioned, fiber channel controllers have a distinct advantage over traditional SCSI controllers in that the fiber network is able to carry data to and from the controller much more rapidly than SCSI.
- If you have the money, consider a separate array controller for each major storage category: OS, databases, log files, and so forth. Doing so will permit SQL Server to maintain parallel data paths and can create a significant performance improvement.
- Purchase controllers with their own onboard RAM cache and battery backup. These features allow the controller to report data as “written,” and allow SQL Server (and Windows) to go about other tasks. The controller then streams data to the physical disks. The battery backup ensures that a loss of power won’t result in a loss of data; when power is restored, the controller will generally write anything left in RAM as soon as the server starts, before Windows even loads.
- Arrays or disks used for SQL Server data shouldn’t contain any other devices. For example, don’t use your array controller to run a tape backup unit, CD-ROM, or other device; dedicate the controller to the task of moving data for SQL Server. Multi-purposing a controller simply divides its attention and reduces the throughput available to SQL Server. In fact, when possible, don’t install tape backups and other devices on a SQL Server computer. Even though you can use a separate controller for these devices, you’re still running the risk of placing an unnecessary burden on the server’s processors.

 If you’re interested in testing your storage performance, download one of the free performance utilities from <http://www.raid5.com>. These utilities can perform a fairly thorough test of raw throughput and let you know how your system is doing. These utilities don’t test SQL Server-specific performance, but gather general, raw disk performance. Microsoft provides a tool named SQLIO, which writes data in 8KB and 64KB blocks, mimicking SQL Server’s own disk usage patterns. This tool is useful to benchmark SQL Server-specific performance.

Aside from the server configuration, a number of SQL Server configuration best practices can help improve performance. For example, high-end installations typically provide separate RAID arrays for each type of data, often storing the OS and SQL Server on a RAID-1 (mirrored) array, data on various RAID 5 (or, more commonly, RAID 10) arrays, log files on RAID 1 arrays, and so forth. SQL Server’s temporary database, Tempdb, is often stored on an independent RAID 5 or RAID 10 array to improve performance, for example; that may seem like overkill, but keep in mind that SQL Server can’t function without Tempdb (meaning you’ll need to allocate plenty of space for it, too). You’ll need to create a configuration that not only provides the best performance but also meets your availability and fault tolerance requirements.

## Summary

In earlier chapters, I introduced you to scale-out concepts and compared scaling out to improved efficiency. I've outlined several scale-out techniques and technologies, including replication, federated databases, distribution partitioned views, and more. I've also discussed Windows Clustering, which can provide a level of server fault tolerance to a scale-out solution. Finally, in this chapter, I discussed high-performance storage, which provides a foundation for better-performing scale-out projects.

High-performance storage is critical to many server operations, not just SQL Server. However, because SQL Server is one of the most disk-intensive applications you can install under Windows, high-performance storage becomes a critical design component of any scale-out solution. Careful attention to the design of your storage subsystem is critical to the success of a scale-out project, as even a well-designed database will suffer if stored on an underperforming storage subsystem.

Although scale-out projects can be complex and require a significant design investment, they are possible and can provide equally significant performance improvements for a variety of large SQL Server applications. In addition, so-called commodity hardware can be used in scale-out solutions to provide a more cost-effective solution, in many cases, than single-server solutions that utilize more expensive, proprietary hardware designs.

## Content Central

[Content Central](#) is your complete source for IT learning. Whether you need the most current information for managing your Windows enterprise, implementing security measures on your network, learning about new development tools for Windows and Linux, or deploying new enterprise software solutions, [Content Central](#) offers the latest instruction on the topics that are most important to the IT professional. Browse our extensive collection of eBooks and video guides and start building your own personal IT library today!

## Download Additional eBooks!

If you found this eBook to be informative, then please visit Content Central and download other eBooks on this topic. If you are not already a registered user of Content Central, please take a moment to register in order to gain free access to other great IT eBooks and video guides. Please visit: <http://www.realtimepublishers.com/contentcentral/>.