

realtimepublishers.com<sup>tm</sup>

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

# Exchange Disaster Recovery & Availability

*Paul Robichaux*



*Jim McBee, technical editor*

---

Chapter 3: Availability Building Blocks: High Availability and Business Continuance .....	42
What Is High Availability, Anyway?.....	42
What Is Business Continuance, Anyway? .....	43
High-Availability and Business Continuance Technologies .....	44
Redundant Arrays of Inexpensive Disks (RAID) .....	44
How RAID works .....	44
RAID: Pros and Cons.....	52
Clustering.....	52
How Clustering Works .....	52
Clustering: Pros and Cons.....	56
Storage Area Networks .....	57
How SANs Work .....	57
SANs: Pros and Cons.....	59
Replication and Failover.....	59
Design Choices .....	60
To Cluster, Or Not To Cluster?.....	60
SANs.....	61
Failover and Failback Design .....	62
Planned vs. Unplanned.....	63
Summary.....	64

## Copyright Statement

© 2005 Realtimedpublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimedpublishers.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 Realtimedpublishers.com, Inc or its web site sponsors. In no event shall Realtimedpublishers.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.

Realtimedpublishers.com and the Realtimedpublishers 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 Realtimedpublishers.com, please contact us via e-mail at [info@realtimedpublishers.com](mailto:info@realtimedpublishers.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 3: Availability Building Blocks: High Availability and Business Continuance

In Chapter 2, you learned about some fundamental technologies that can be used for disaster recovery. Most businesses are more interested in *preventing* disasters than in recovering from them. There are several technologies that can potentially raise the availability and service quality of your messaging systems, and the infrastructure on which they depend. This chapter will examine some of these technologies and see how you might apply them in your Exchange infrastructure.

### What Is High Availability, Anyway?

Let's start the discussion of what high availability and business continuance means with a quote from Marcus' and Stern's *Blueprints for High Availability* (Wiley):

*High availability once referred to a fairly specific range of system configurations, usually involving two computers that monitor and protect each other. During the last few years, however, it has lost much of its original meaning; vendors and users have co-opted the term to mean whatever they want it to mean.*

On its face, this sounds about right; different hardware and software vendors have vastly different definitions of what "high availability" actually means. So do administrators; if you think back to preceding chapters' discussions of RTO and RPO, it should be fairly clear that, say, the New York Stock Exchange and a local doctor's office probably have different definitions of how much uptime is required to be "highly available." For the purposes of this guide, we're really more interested in talking about *resiliency*, or the ability of a system to survive failures or malfunctions in individual components. In general, you can fairly say that a highly resilient system will deliver high availability because its resiliency increases its resistance to unplanned downtime.

For example, suppose that the magic number that defines a high-availability system is an uptime level, excluding planned downtime, of 99.9 percent or higher. 99.91 percent and you're in; 99.0 percent and you're out (after all, there's almost a 3-day difference between 99.0 percent and 99.9 percent!). This level might seem low, but it's a realistic starting point. The goal in this chapter, then, will be to identify technologies that may help you provide at least 99.9 percent of planned uptime in your environment. Most of the methods that you can use to provide high availability for your Exchange infrastructure are technical in nature, and we'll be discussing them in the rest of this chapter.

## What Is Business Continuity, Anyway?

The definition of *business continuity* is a little less amorphous than the one for high availability—business continuity is the ability to keep critical business operations going during an outage or disaster *and the ensuing recovery*. The variables in this definition involve time and money: how long does it take you to resume operations, and how much money does it cost to do so? The preceding chapter talked about RTO and RPO as ways to specify how long recovery operations may take; you can think of a switch to business continuity-based operations as another type of RTO. Although it's not a standard industry term, this chapter will use the term *continuity time objective* (CTO) to represent the time it takes to fail over from conventional operations to business continuity operations.

The methods by which you can implement business continuity span a wide continuum of cost and effort. They tend to fall into a few distinct categories:

- You can keep offsite backups, then move, carry, or ship them to a remote office where you already have staff and hardware available. This option is the least costly, but it tends to have a higher CTO because you can't resume normal operations until you get your backups to the remote site and get them loaded.
- You can use software or hardware replication to copy your critical data to an alternative server at a remote office. Although adding replication costs more (*a lot* more if you're using hardware replication), this method cuts the CTO because the data you need at the remote site will be up to date already. As a bonus, many replication solutions include scripts or tools to automate the process of failing operations (and clients) over to the continuity system.
- You can use either of the previous approaches, but with a third-party business continuity provider such as SunGard or IBM Global Services. How much does this option cost? That's up to you and your sales representative! The prices these firms charge vary widely according to your CTO and what optional services you want included, so there's no way to give even an estimate here—except to say that, as with most other IT outsourcing services, you may find in the end that the cost outweighs the actual value.
- You can use dispersed clustering (also known as stretched clusters, or by EMC's trademarked term “geo-clusters”) to widely separate your cluster nodes. Dispersed clusters are, technically speaking, very cool. They are also expensive, and they require a great deal of specialized knowledge to operate.
- You can maintain your own separate parallel operation, using replication to keep multiple intra-day copies of your data at a fully staffed, ready-to-go remote site. This method is essentially what some large operations, such as the New York Stock Exchange and the US Space Command, use to ensure that their CTO is as short as possible. However, as you might expect, this option is usually the most expensive because it essentially involves duplicating every aspect of your computing environment *and* the people who maintain and operate it.

The missing ingredient in all of these categories is the *process* you use to implement business continuance. With high availability, one of the key ideas is that you can buy high-availability technologies and get their benefits without necessarily changing your processes. With business continuance, you're buying into a wholesale set of process changes as part of the entry cost; without making those changes, you won't be able to realize the full benefit of your business continuance investment. As the next chapter will explore, improving your high availability and business continuance processes can often do more to improve your uptime, and shorten your CTO, than anything else.

## High-Availability and Business Continuance Technologies

The technologies available to implement high-availability and business continuance solutions are pretty fascinating. They also tend to be complex and expensive, which is to be expected given what they do. This section will examine critical technologies and discuss their applicability to high availability and business continuance. Chapter 2 covered the basic technologies that you can use to improve the disaster recovery capabilities of your organization; this section will examine the technologies that you can apply to provide better high-availability and business continuance support.

### ***Redundant Arrays of Inexpensive Disks (RAID)***

At the time of its invention in the early 1980s, the idea behind RAID was pretty radical: use a specialized controller and a set of relatively cheap, dumb disks to provide data protection—and thus higher availability—without the expensive, support-intensive infrastructure then common in mainframe and minicomputer implementations. As you can imagine, this idea caught on quickly, and now RAID is common in even low-end server hardware from major server manufacturers. A variety of third parties sell RAID controller cards that you can add to existing systems, and server versions of Windows include their own software RAID implementation that you can use to provide some types of RAID in lieu of, or as a supplement to, additional hardware.

### **How RAID works**

RAID is a familiar acronym to most systems administrators, but a surprising number of people don't know how it actually works or where the implementation tradeoffs are. The first important thing to understand is that there are several different RAID levels, each of which trades off capacity utilization, performance, resiliency, recoverability, and cost. There is not really a linear relationship between these tradeoffs and their associated numbers, especially because there are a number of levels that don't offer enough performance, cost reduction, or resiliency to make them worth implementing (heard of RAID-4 on a server lately?) However, some RAID levels are better suited for Exchange than others. In the following discussion, let's assume that we're talking about a single logical volume built with the RAID level under discussion.

### RAID-0/Striping

Let's start with the first RAID level, which is also one of the simplest: RAID-0, or striping (see Figure 3.1). The idea here is simple: the logical volume is built so that a portion of its data is stored on each physical disk in the volume set. The *stripe size* (a term you'll see again throughout this book) refers to the amount of data written as a chunk on each physical disk. Data to be written are evenly distributed across the physical disks in the stripe set. You must have at least two disks to have a RAID-0 volume, but you can add any number of additional disks. Let's assume that you have three disks with a 64Kb stripe size (the default for most Windows-based RAID implementations). A 62Kb chunk of data would be written as a single chunk on one disk. A 250Kb chunk would be written as a series of six 64Kb chunks, two on each of the disks in the stripe set. The performance advantage is obvious: as the six writes required are distributed across three physical disks, writing is (roughly) three times faster than writing to a single disk. In addition, RAID-0 provides effective utilization, because no space is wasted on parity or redundancy. Striping is simple to implement, too.

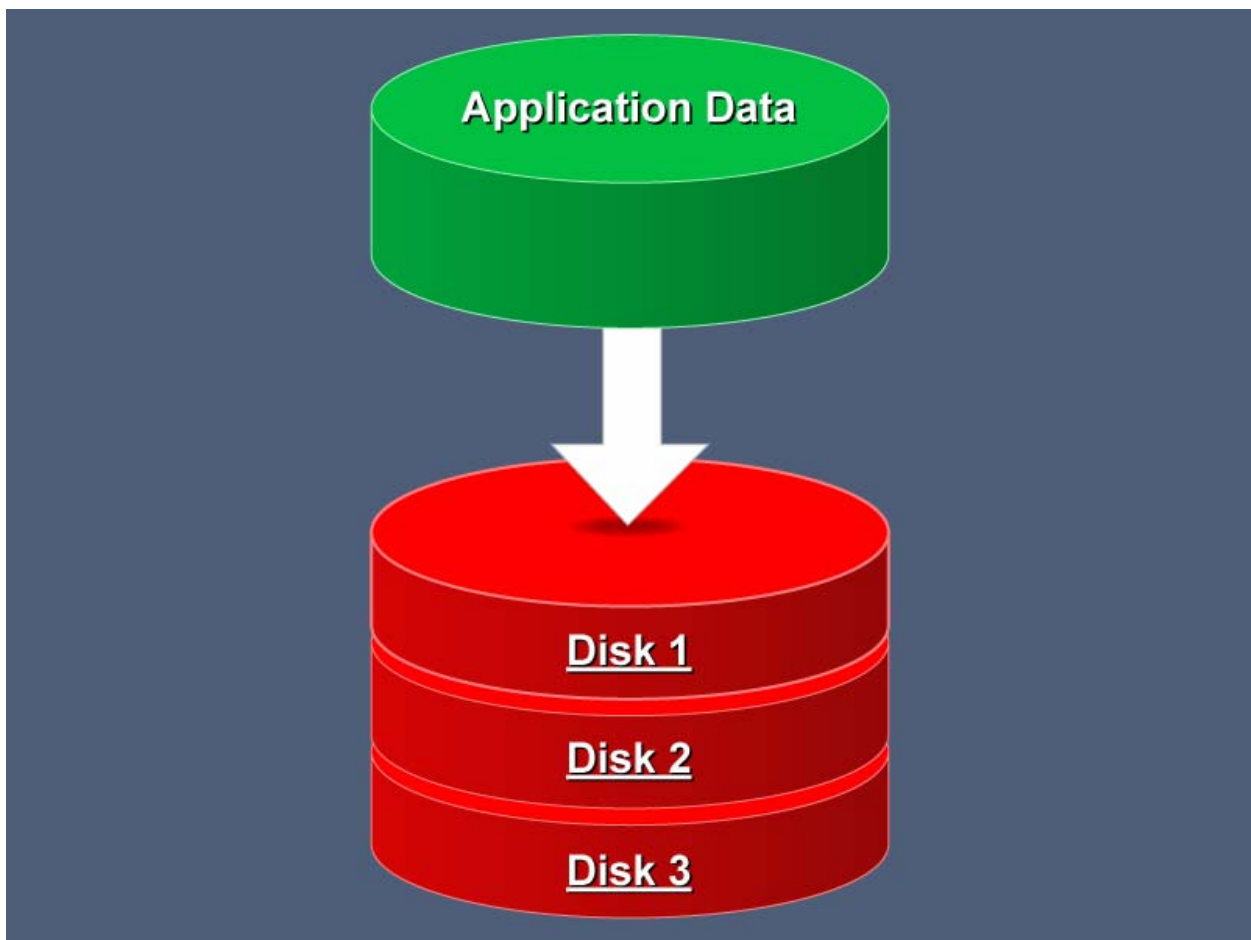


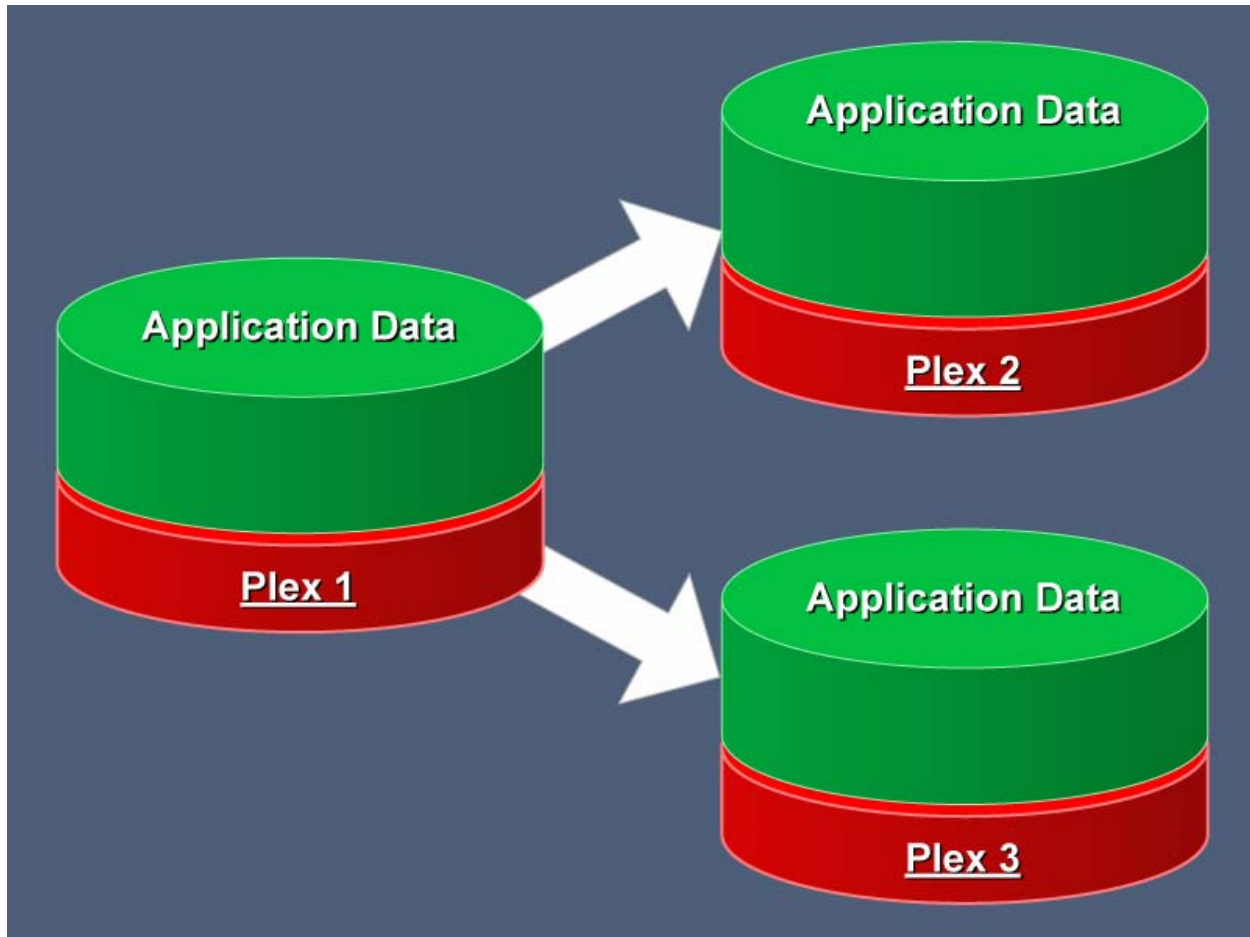
Figure 3.1: RAID-0 distributes data across multiple disks.

Of course, there is one glaring problem: RAID-0 doesn't provide *any* redundancy or resiliency. If one of the drives in the stripe set fails, you lose access to *all* the data in the stripe set. Thus, it's best used to provide storage for services that hold data of transient value. In the broader industry, you'll often find RAID-0 implementations used for video editing and streaming; for Exchange, you're likely to see this configuration used only for SMTP queues, and even then only on fairly busy servers. Its lack of redundancy makes RAID-0 wildly inappropriate for use with Exchange mailbox databases, even though the write performance and efficient space utilization would be most welcome.

#### **RAID-1/Mirroring**

Striping doesn't offer any redundancy, but RAID-1 (better known as mirroring) offers it in capital letters. As Figure 3.2 shows, RAID-1 mirrors data by taking data written to a primary volume, or *plex*, and writing the same data to additional physical volumes. (Windows, and most hardware RAID implementations, limit you to two plexes, so that is what we'll discuss here.) The plexes in the mirror set are treated as a single logical volume. This setup introduces a performance tradeoff; to write a given block of data, you have to do twice as many writes as with a single disk (which may or may not make a real performance difference, depending on how fast the disks are), but you can read it back twice as fast. Thus, at least from a performance standpoint, mirroring works better for applications with more reads than writes, and for I/O patterns that involve more sequential than random accesses. (You can mitigate this impact by putting the two mirror halves on separate disk controllers, provided that your RAID implementation allows doing so, but that adds expense and complexity.)





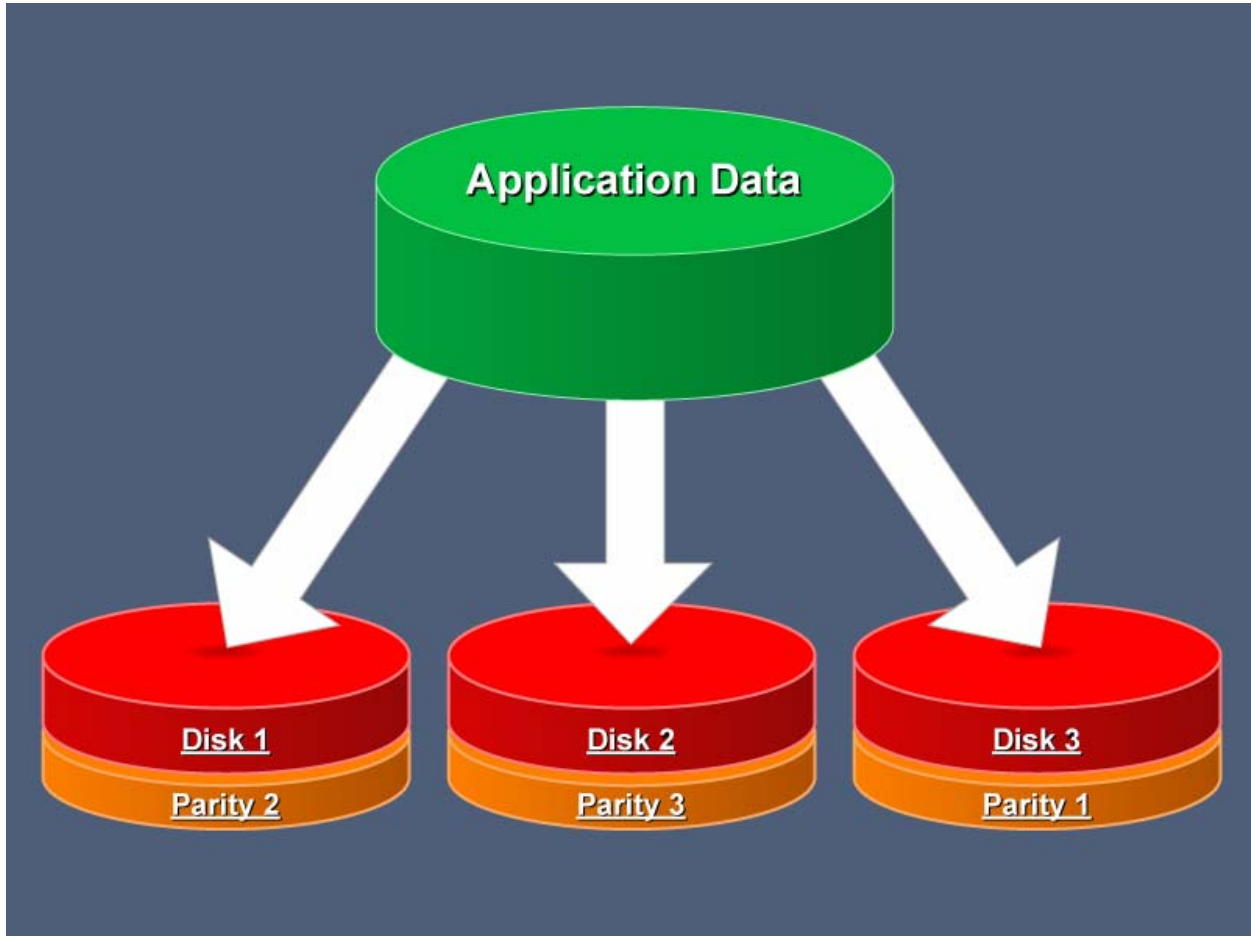
**Figure 3.2:** RAID-1 mirrors data from a primary volume onto one or more secondary volumes.

Of course, the resiliency benefits of RAID-1 are more often cited as justification for using it. If you lose one of the disks in a RAID-1 set, you can replace it and easily rebuild the mirror set by copying data from the remaining disk. Most controllers, and the Windows software RAID implementation, do so automatically. This action makes RAID-1 well suited for storing Exchange transaction logs, which are always written to sequentially. In fact, Microsoft recommends the use of mirroring for transaction logs and the system boot volume.

Mirroring does have a significant drawback: it provides resiliency but not redundancy. If the data you write to one disk is corrupt or incorrect (say, because you accidentally ran a file-based antivirus scanner on your transaction logs), the corrupted data will automatically be mirrored to the other disk. Mirroring systems almost always involve disks in close physical proximity, too, so if you lose one of the disks, you're likely to lose the other. That's why replication-based solutions are so popular; they give you an easy way to combine the resiliency of mirroring with remote redundancy.

**RAID-5/Striping with Parity**

Striping offers great performance, but it's fragile because the loss of any disk of the stripe set means you lose access to all the data therein. RAID-5, also known as *striping with parity*, attempts to solve this problem by calculating redundancy data when data is written, then distributing it across the disks in the logical volume, as Figure 3.3 shows.



**Figure 3.3:** RAID-5 mixes data and parity to provide fast I/O and good recoverability.

RAID-5 sets must have at least three identically sized physical drives. Unlike RAID-0, some of the space of the physical drives in a RAID-5 set is consumed for redundancy; this consumption has the effect of reducing the available storage by an amount equivalent to the capacity of a single disk. For example, if you build a RAID-5 set of 5 400GB drives, the actual usable capacity is 1.6TB—the other 400GB is used to hold redundancy data. However, the parity data is distributed across *all* the disks in the array; there's no “parity disk” because putting the redundancy data on a single disk would introduce a single point of failure for the entire array.

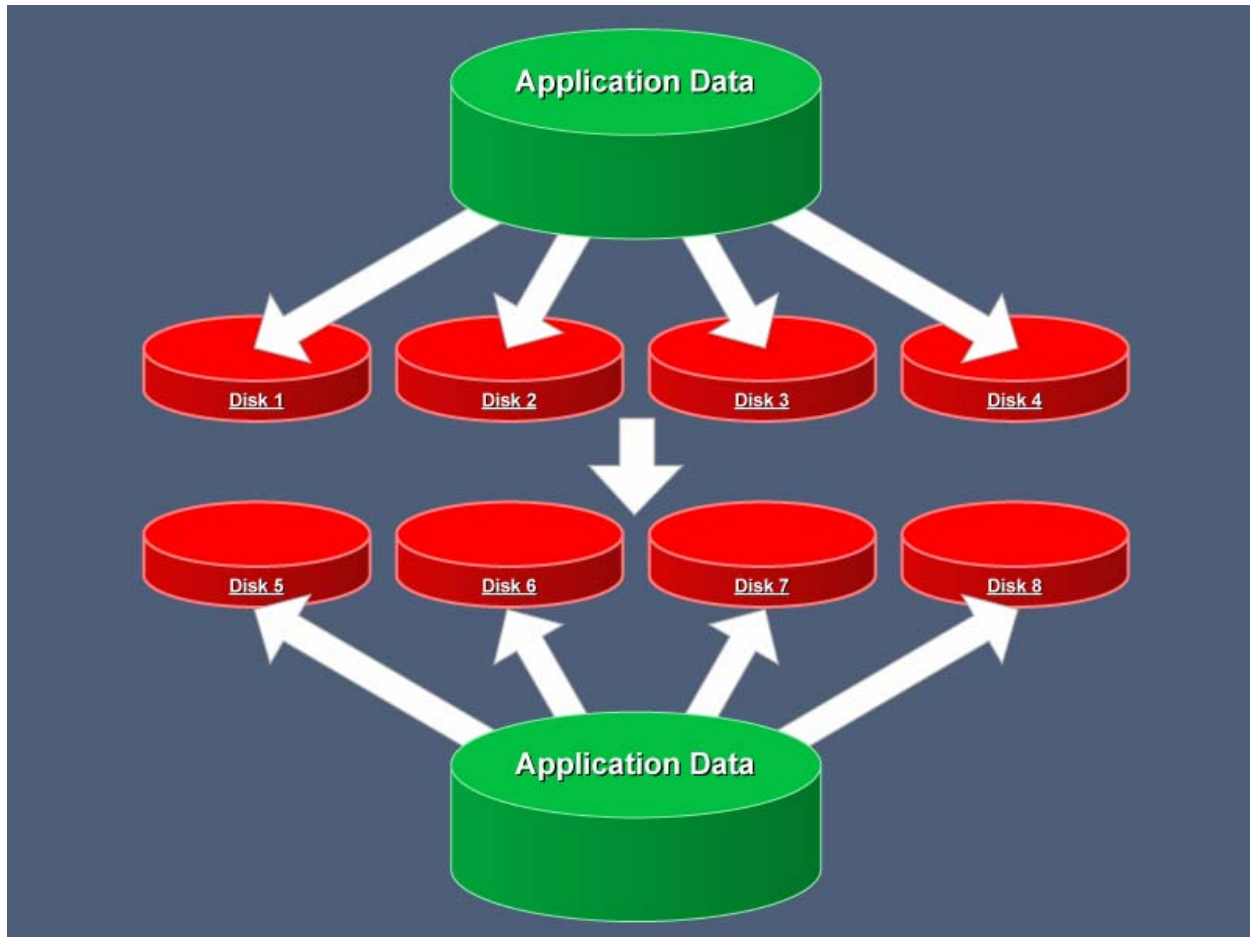
The parity calculation is a simple exclusive-OR (XOR) operation of all the blocks in a stripe set. The use of XOR allows any single disk block in a group to be reconstructed from the others; for this reason, RAID-5 arrays can tolerate the loss of a single drive without losing data. It's important to remember, though, that during the time that the missing drive's contents are being reconstituted, the array's performance will be significantly slower than usual.

Parity is generated when a write occurs. For example, say you're writing a 300KB chunk of data on a five-disk RAID-5 array with a 64KB stripe size. Thus, there are 5 blocks of data, so each drive gets a single chunk, and one drive (chosen by the controller) gets the parity block for that set of data blocks. If any of the data blocks are lost, the remaining blocks and the parity block can be combined to reconstruct the bytes in the missing data block. It's critical that the parity and data be kept consistent. If the parity block doesn't match its corresponding data blocks, it won't be possible to recover those blocks. For that reason, the parity has to be recalculated any time a single block in a data set changes—and that means that the existing blocks have to be read so that the newly written data will have correct parity information. This read-modify-write cycle is handled invisibly by the controller, which is why server RAID controllers typically include a battery-backed non-volatile RAM (NVRAM) cache on the controller card.

Of course, the most common reason for a reconstruction is because a disk has failed. Most hardware RAID-5 implementations make provisions for spare drives (*hot spares*) that can be automatically or manually used to replace a failed disk. One thing to watch out for in RAID-5 arrays is that the array data is vulnerable to a *second* disk failure while the array is rebuilding—most RAID-5 controllers can only handle failure of a single disk. When are disks most likely to fail? When they're being exercised strenuously—which is just what happens during an array rebuild. To compound the problem, if your array is set to automatically swap in a hot spare disk to replace a failed one and you don't notice that this has happened, when you lose a second drive, you may find your array suddenly useless! Be sure to keep a close eye on the status of your hot spares.

#### **RAID-0+1/Mirrored Stripes**

RAID-0+1 combines mirroring and striping; data on the logical volume is striped across multiple physical disks to make an ordinary stripe set. That stripe set is then mirrored to another set of disks, giving a configuration like the one that Figure 3.4 shows. RAID-0+1 has the advantage of providing terrific I/O throughput, as both reads and writes are distributed across multiple spindles. In fact, with a suitable controller, reads can be up to twice as fast as a single mirror set because the controller can simultaneously read from both sides of the mirror.



**Figure 3.4:** RAID-0+1 combines mirroring and striping.

However, there's a problem with RAID-0+1: it offers the same fault tolerance as RAID-5, but it's much more expensive. You can lose a single drive from either half of the mirror, at which point *all the data on that side of the mirror becomes unusable*. If you lose one drive from side A, then lose one drive from side B before side A is completely rebuilt, you've just lost the whole array! Worse still, RAID-0+1 is very expensive because you have to build out both sides of the mirror. It's commonly found in applications such as prepress imaging servers, where performance is more important than anything else, but it's not really suitable for use with Exchange.

#### **RAID-10/Striped Mirrors**

Although it's easy to do, don't confuse RAID-0+1 with RAID-1+0 (or RAID-10). The only thing they have in common is the presence of the numbers "1" and "0" in their names. RAID-0+1 is a mirrored pair of stripe sets, and RAID-10 is a striped pair of mirrors. Figure 3.5 shows what this design looks like; admittedly, it's a bit confusing, but the basic idea is simple enough. Instead of using individual disks in a stripe set, RAID-10 uses *mirrored pairs* of disks. This setup provides the effective I/O performance of striping with the resiliency of mirroring.

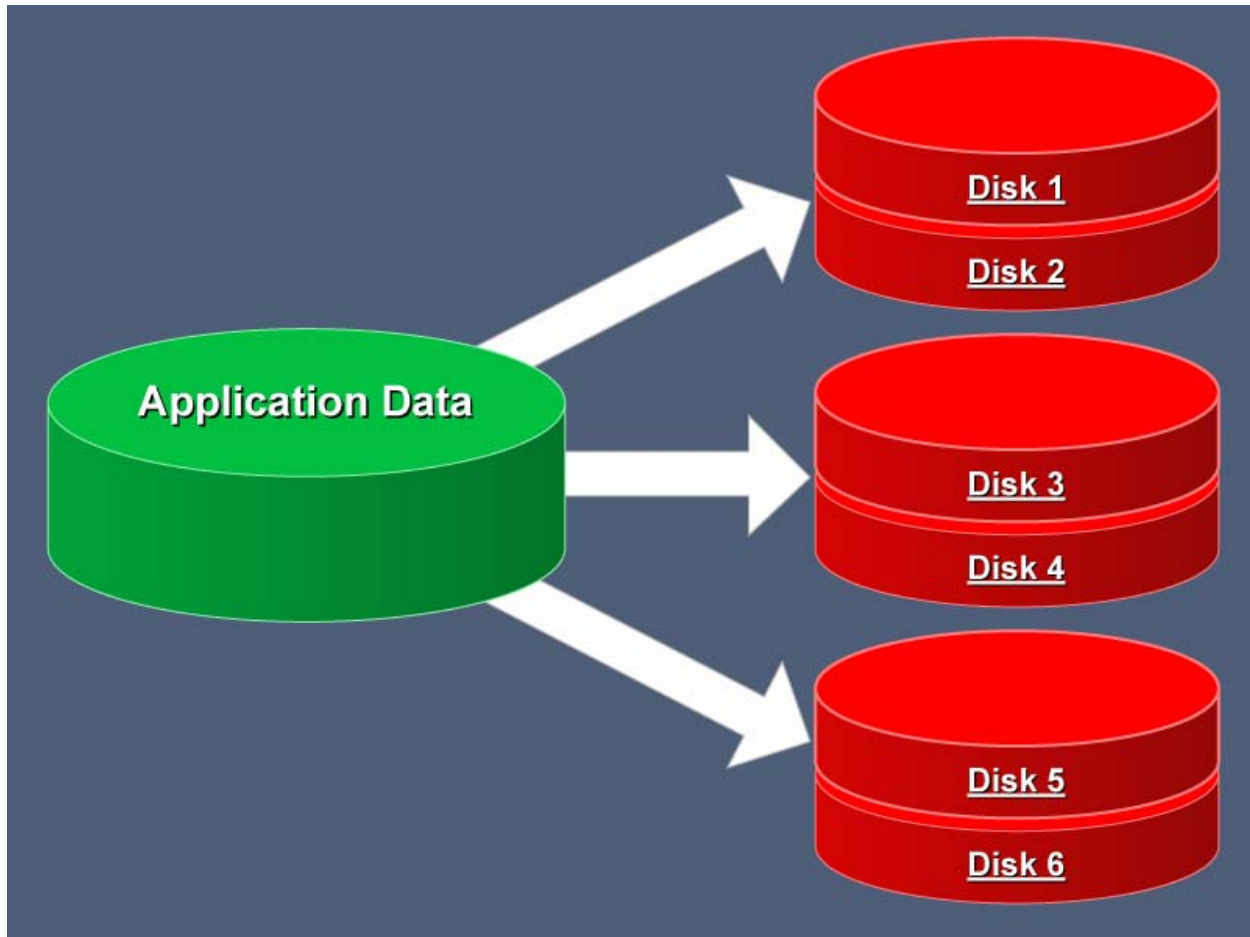


Figure 3.5: RAID-10 combines mirroring and striping, but in a different manner than RAID-0+1.

If you lose a single disk from a portion of the stripe set, you're still OK because the other disk is protecting you against losing the entire stripe set. In fact, you can lose one disk from *each* portion of the stripe set and still continue normal operations; the only event that can cause data loss is a failure of *both* disks in a stripe portion, or a failure of the controller itself. Because both of these are unlikely, RAID-10 gives excellent protection.

There's an associated cost, of course; let's say that you wanted to build a 1TB array for your Exchange data. You could use 5 250GB disks in a RAID-5 array, but to get the same storage in a RAID-10 array you would need 8 disks, arranged as four pairs. In other words, you would have to spend 62 percent more on disks to get the same amount of storage. However, because RAID-10 provides better write and read performance than RAID-5, with equal or better resiliency, it's very well suited for Exchange provided you can absorb the extra disk cost. Microsoft recommends the use of RAID-10 for Exchange databases when the individual disks in use are larger than 18GB, although I'm not sure where they got the 18GB limit.

## RAID: Pros and Cons

Most of the time, there are legitimate tradeoffs for high-availability, business continuance, and disaster recovery technologies. Such is not really the case with RAID; the question isn't whether you should be using RAID, but rather which RAID level you should be using. In general, I don't recommend the use of low-end RAID with Exchange, which includes Windows' built-in RAID software and cheap PCI cards that provide RAID-like mirroring and striping. For a RAID implementation to be useful *and* safe, it needs to have a controller with onboard cache, and that cache needs to have a battery backup. (Bonus points if you can remove the cache SIMM and put it into another identical controller; that helps insulate you against controller failures.) Table 3.1 summarizes recommendations for the use of various RAID levels.

RAID Level	Recommended Use
0 (striping)	Use for striping the SMTP queue volume on busy gateway machines; not appropriate for other Exchange use
1 (mirroring)	Microsoft recommends using mirroring for transaction log volumes and the system boot volume; don't use it for databases
5 (striping + parity)	Commonly used for databases because it provides good resiliency and redundancy; worse performance than RAID-10, but somewhat cheaper
0+1 (mirrored stripes)	Don't use for Exchange, and don't confuse it with RAID-1+0
1+0 (striped mirrors)	Great for use with Exchange databases; excellent performance, resiliency, and redundancy, but costs more than RAID-5

**Table 3.1: Recommended use of RAID levels.**

## Clustering

Before we begin a discussion of clustering, a vocabulary clarification: please don't confuse a *server cluster*, which combines multiple independent computers to provide failover capability, with a *network load-balancing cluster*, which distributes incoming traffic between multiple machines. Microsoft's network load-balancing team persists in using the term "cluster" to refer to an aggregate set of NLB machines, and that's not what we're talking about here.

## How Clustering Works

With that said, what *is* a cluster? Microsoft defines it as a cooperating group of nodes that share a group of cluster resources. There are several types of resources, ranging from network names and addresses to Exchange virtual servers. The cluster software, which runs on each node, takes care of moving resources between nodes as necessary when a node fails or an administrator requests failover. The key Exchange resource we're interested in is called an *Exchange Virtual Server* (EVS); you can think of it as a complete bundle of everything clients need to get access to mail data. Each EVS contains, at a minimum, a network name, an IP address, one or more physical disks, and an Exchange system attendant resource. Depending on what the server's doing, the system attendant may create other resource instances, such as mailbox or public folder databases, SMTP and HTTP servers, a message transfer agent, and a routing engine.

Individual resources can move from physical server to physical server, but the primary method of moving resources is to group them into *cluster groups*. The EVS is a type of cluster group; clients connect to an EVS and treat it just like a physical Exchange server, except that at any time the EVS can be moved to another physical node in the cluster.

Windows implements what's known as a *shared-nothing* cluster: any cluster node can potentially own any of the resources in the cluster group, but each resource can only be owned by one server at a time. For example, if you've created a logical disk volume for a storage group and its transaction logs, that volume can only be accessed by one cluster node at a time. When a resource transfers to another node, we call that a *failover*. Failovers may be triggered by manual administrator action or via the cluster software when it notices a failure.

Failovers are coordinated by the use of a *quorum* resource, which is basically a disk volume that contains state information about which nodes are in the cluster and who owns which resources. In the event of a failure, a surviving cluster node can take control of the quorum resource and use it to orchestrate a failover.

Figure 3.6 shows a typical 4-node cluster. Microsoft recommends keeping at least one passive node in the cluster, so the cluster shown in the figure has active EVSs on nodes 1, 2, and 3. If any of these nodes fails, its EVS will automatically fail over to node 4. In fact, node 4 could potentially accept the EVSs from all three of the other nodes, although performance would suffer quite a bit.

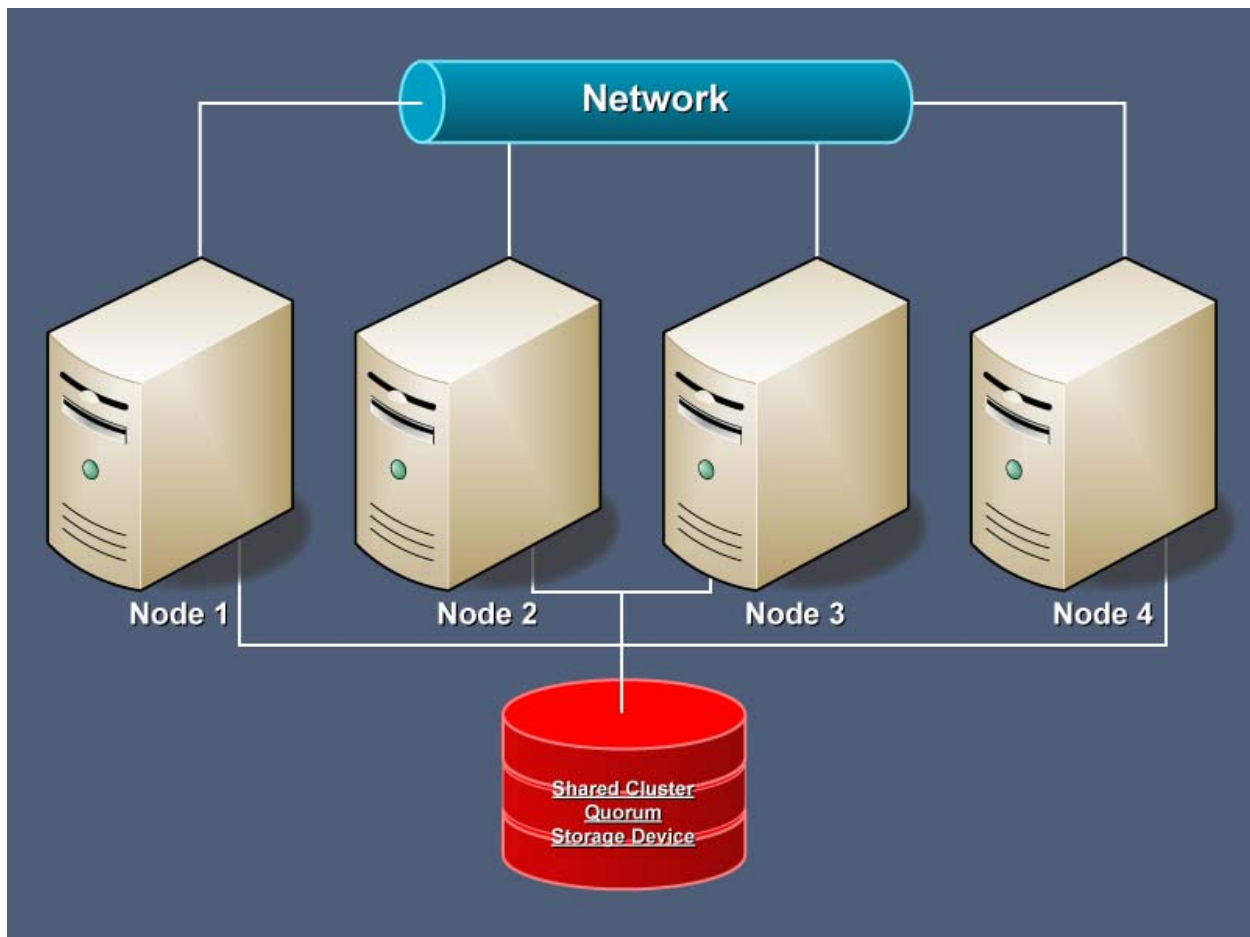
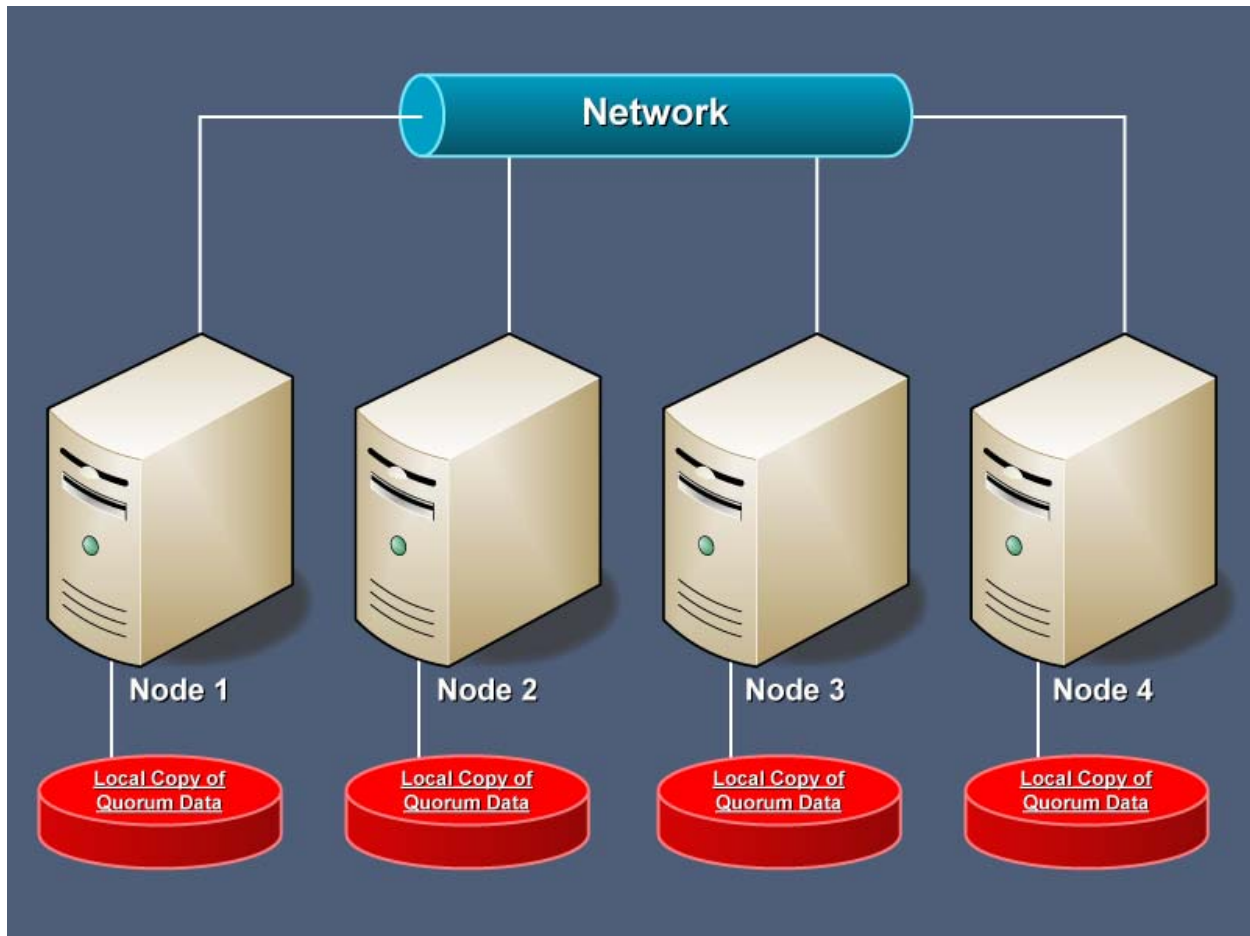


Figure 3.6: A conventional server cluster relies on the shared quorum disk.

This seemingly ordinary design hides a serious problem: if the storage system that maintains the quorum fails, or if the volume on which the quorum resides is damaged, the entire cluster can fail because it becomes very difficult for the remaining nodes to determine which node is in charge. In his landmark book *In Search of Clusters*, Gregory Pfister dubbed this occurrence “split-brain syndrome,” and it’s definitely something to be avoided. The best way to solve this problem is to not *have* a single quorum. Instead, use *majority node set (MNS)* clusters decentralize the quorum so that each cluster node has its own local copy, which is synchronized with other copies, as Figure 3.7 shows. (MNS clusters are still relatively rare because the addition of quorum synchronization makes them more complicated than ordinary clusters.)



**Figure 3.7:** MNS clusters don’t use a single shared quorum.

Both of these configurations require that the OS, and applications running on it, be aware that they live in a cluster. Of course, Exchange 2000 and Exchange Server 2003 are both cluster-aware, so if you install them properly on a cluster node, they can take advantage of the cluster’s existence. Exchange 2000 doesn’t support the use of MNS quorums, and its failover mechanism is considerably slower than the Exchange Server 2003 version.



### Cluster Failover

Speaking of failover: here's how it works. Manual failovers can be initiated by the administrator at any time via the Windows cluster management MMC snap-in. When you move a resource from one node to another, that action constitutes a failover. You might do so to allow for planned maintenance or hardware upgrades, for example. The more interesting situation is that of automatic failover, because that's the primary feature that clustering offers in exchange for its complexity and expense.

Each cluster node is connected to the other nodes via a dedicated network interconnect. This interconnect is used to send (and receive) UDP-based "heartbeat" notifications that indicate that each cluster node is still powered up and participating in the cluster. If the notifications from a given node stop, the cluster service assumes that the node has failed, and another node will seize its resources and begin offering service with them. (You can see the obvious problem if the "failed" node isn't really dead!) The allowed targets for a failover are set by a failover policy, so you can specify which individual nodes should pick up work from other failed nodes.

There is a second layer of failover awareness built-in to Microsoft's clustering solution. In addition to the machine-level heartbeat system, which indicates whether a node is awake and connected to the network, the cluster service provides for a way to monitor the health of an individual resource: the `IsAlive` call. For Exchange, individual resources are checked with `IsAlive` every 10 seconds; if an `IsAlive` call fails, the associated resource can be failed over. Different services have their "aliveness" checked in different ways; for the Exchange system attendant, routing engine, and MTA stack service, the cluster resource manager queries the Windows service control manager to determine whether the services are running; for the information store, the resource manager makes an RPC request basically asking "are you alive?" SMTP, POP, IMAP, and HTTP are tested by making queries to their respective ports. If one of these calls fails, the resource manager will decide that the associated resource has failed, and it will initiate a failover of that resource *and* any resources it owns.

You might find it interesting to know what takes up the largest percentage of time during a failover. In almost all cases, the most time-consuming part of the EVS failover is replaying the transaction logs for the storage groups on the EVS that failed over. The actual failover operation itself can be very quick—less than a minute in some cases—but having to replay large numbers of log files will kill your failover performance. (To fix this, you can adjust the number of log files kept on the servers, but doing so is only advisable if you know what you're doing in great detail).

One additional type of clustering deserves mention: geographically dispersed or "stretched" clusters. These look like ordinary clusters, except that the distance between nodes can be kilometers instead of mere meters. These clusters are typically implemented by using dedicated fiber-optic network links that provide extremely high bandwidths and low latencies (in fact, there's a practical limit of about 100Km between cluster nodes, imposed by the time it takes light to travel that distance). Remarkably enough, Exchange doesn't have to know, or care, that the underlying cluster is dispersed; as long as I/O latency stays low enough (that is, as long as disk writes finish in less than about 500ms) the cluster can operate normally. However, as the distance increases, so does the latency, which means the number of users you can support per node *decreases*. Between that restriction, and the insanely high cost of the communications infrastructure required, these clusters tend to be the province of organizations with an extremely large IT budgets and extremely high and well-justified uptime requirements.

## Clustering: Pros and Cons

Clustering is often presented as a magic bullet for high availability. It's not. Having said that, there *are* some things that clustering does very well. Appropriately sized, planned, and deployed, Exchange clusters give you:

- *Protection against single points of failure* on individual cluster nodes. For example, if the motherboard fails in one of your cluster nodes, its EVS will fail over to another node and you'll continue working normally.
- *A good combination of scalability and resiliency*. Because you can put as many as 8 nodes in an Exchange Server 2003 cluster, with the right hardware, you can host thousands of users with fairly robust protection against failures.
- *Seamless failover*. In a properly administered cluster, users need never know that anything untoward has happened when a failover occurs. This benefit is the number one reason clusters are widely deployed; most administrators love the idea that their users can keep working during a failure without flooding the Help desk and IT staff with trouble reports.
- *Invisible downtime*. Combining Outlook 2003's cached Exchange mode with clustering gives you more or less invisible downtime—if you have a failover, Outlook insulates users from noticing the disruption.
- *A very useful way to deploy rolling upgrades or patches*, even during normal business hours. This functionality is a very useful capability, not least because it gives you a way to deploy Windows and Exchange hotfixes and service packs with minimal interruption of service.

Clustering has some pretty significant negatives, too, that make it inappropriate for many uses. The following list highlights these drawbacks:

- *It's expensive*. A 4-node cluster can cost significantly more than a deployment using identical hardware as standalone servers. Why? Shared storage is an obvious factor, because it tends to be considerably more expensive than normal direct-attached storage. You also need to add on the cost of the cluster interconnects themselves, plus the cost of cluster-aware versions of whatever antivirus, antispam, and backup tools you're using. Then factor in the inconvenient reality: your 4-node cluster can handle only the same amount of active users as *three* similarly configured servers, so you're essentially paying for an extra server (and its power, and its licenses, and so on) as a protective measure. That's not always a good investment.
- *It's demanding*. You really have to know your stuff to set up and run an Exchange cluster. Exchange itself pretty much works the same way on clustered and unclustered systems, except that some components (such as the Exchange Intelligent Message Filter) aren't cluster-aware. In particular, shared storage systems based on Fibre Channel or shared SCSI tend to be somewhat finicky by comparison to DAS, and the skills and knowledge needed to manage shared storage take time to develop.

- *It's not perfect protection.* Clustering doesn't protect you against many common failure modes. For example, it doesn't do much to protect against administrator errors (the largest single source of cluster failures, in fact), and—barring the use of geographically dispersed clusters—it won't protect you against physical damage to the cluster nodes themselves, nor does it do anything to protect you against data corruption.

The key to deciding whether clustering is appropriate for your environment is simple. Ask yourself two questions:

- What does downtime cost you
- How much downtime would clustering prevent in your environment?

A general recommendation is that, unless you can reliably hit 99 percent uptime or higher *without* clustering, you should probably work on raising your uptime through other methods before implementing clustering.

### **Storage Area Networks**

Storage area networks (SANs) are designed to make storage fungible. With DAS, disks are tied to individual servers, so it's not feasible to reallocate storage among machines. In addition, the capacity of individual servers is limited to the number of directly attachable storage devices. For example, in a typical 2U rack-mount server, you can install four to six internal drives—if you happen to need more storage than that arrangement gives you, you'll have to add an external enclosure. In addition, if you have one server with excess disk space, you can't easily move it to another server where you need additional space, and you have to back up and maintain it on each individual server.

SANs offer a remedy to these shortcomings. Instead of attaching storage to individual computers, you put lots of physical disks into a SAN, which can then be accessed by multiple machines at once. The SAN controller and enclosure provide services including hardware RAID and the ability to reassign logical volumes from one host to another. In theory, by using a SAN, you can pool all your storage and allocate it to the applications that need it; if you run low on Exchange storage space, you can easily reallocate physical disks from the storage pool and dynamically extend the Exchange volume without any downtime. In addition, by their nature, SANs are intended to be shared, which makes them well-suited for deployment with clustering technology.

### **How SANs Work**

Storage management requires a fair amount of intelligence; in DAS systems, or even in basic shared storage systems, this intelligence is embedded in the disk controllers of the local systems. SANs move that intelligence to a centralized storage controller. Individual computers that attach to the SAN are actually communicating with the SAN storage controller, and the device (known as a *host bus adapter*—HBA) that attaches hosts to the SAN can be thought of as a sort of specialized network card. The HBA, its BIOS, and its drivers are responsible for presenting SAN volumes to Windows so that they look like any other kind of disk volume. As far as Windows is concerned, a SAN volume should look and work just like any other; this seamlessness enables some interesting capabilities, including the ability to boot servers directly from a SAN volume (a process known as *SAN booting*).

There are two primary interconnect methods used for SANs:

- Fibre Channel is an arbitrated command protocol that can be carried over optical fiber or copper cabling. Fibre Channel SANs can be constructed using point-to-point, loop, or fabric topologies; fabrics are the most flexible, but they require the use of specialized Fibre Channel switches to interconnect SAN hosts and devices. The switch's job is to enable any two devices to have a direct connection, in exactly the same way that an Ethernet switch enables full-speed communications between two devices on a shared medium.
- iSCSI essentially encapsulates SCSI commands in TCP/IP packets; instead of using Fibre Channel to interconnect SAN devices and hosts, every device in an iSCSI SAN appears just to be an ordinary network device. Although in theory you could slap iSCSI devices on your existing network, for security and performance reasons, all major iSCSI vendors recommend that you use a dedicated network for iSCSI traffic. Having said that, Gigabit Ethernet HBAs, switches, and so on tend to be much less expensive than the corresponding Fibre Channel components.

In either case, the SAN controller is in charge of marshaling data for the disks in the SAN and presenting those disks as an appropriate set of logical volumes. The controller is also responsible for accepting arbitration requests to ensure that only one computer at a time is attached to a given logical volume. This single ownership is absolutely critical to using clustering with SANs because the ability to safely change the ownership of a logical volume from one host to another is fundamental to failover.

Of course, the architecture I've just described has one important flaw: it doesn't provide any additional redundancy beyond what you could implement with an effective RAID controller and DAS; in fact, it's *less* redundant because multiple servers share a common storage pool. The solution to this problem is fairly straightforward: add more redundancy. Current SAN hardware has multiple controllers, and it's common to put multiple HBAs in each host with each HBA connected to a different switch. This setup eliminates the single points of failure of the HBA and SAN switch (notice that you can design a topology such as this using either Fibre Channel or iSCSI). Windows Server 2003 (WS2K3) includes multi-path SAN drivers that can dynamically fail over work between HBAs in the same machine. For extra protection, you could add a second SAN and replicate data to it using either hardware or software replication.

It's important to note some of the differences between Fibre Channel and iSCSI SANs. In brief, Fibre Channel SANs tend to offer better performance at a higher total cost. Currently, top-of-the-line Fibre Channel hardware can run at 4Gbps, which is roughly four times faster than Gigabit Ethernet, the best-performing medium available for iSCSI. As you might expect, this difference has a significant impact on observed SAN throughput; it doesn't hurt that many Fibre Channel SANs use purpose-built disk drives that use integrated Fibre Channel interfaces instead of more conventional SCSI or ATA flavors.

This is not to say that iSCSI SANs don't perform well themselves; by dedicating Gigabit Ethernet connections between each host and the iSCSI SAN device, and by adding an appropriate number of spindles, you can get excellent performance for considerably less money than an equivalently sized Fibre Channel array.

## SANs: Pros and Cons

SANs offer an exciting potential: put all your storage on a SAN, and any server can access it, with easy manageability and the ability to quickly reallocate storage resources to wherever they're needed. With smart SAN management software, you can even watch the SAN to see which physical disks are handling the largest volume of disk I/O requests, then spread their data to other disks to ease the load. Along with that, a properly designed, implemented, and managed SAN can deliver terrific performance, driven by high disk spindle counts and extremely efficient disk controllers. In addition, they offer a high degree of redundancy and excellent resiliency.

However, these advantages come at a price. SANs aren't for the faint of heart or the tight of purse. If you want the flexibility and redundancy they offer, you're going to have to pay for it, one way or another. This cost takes two primary forms. First is acquisition cost. SANs cost much more than equivalently sized DAS systems. The controllers, enclosures, and so on are just part of the difference; because HBAs are required (and in many designs, so are switches) there's a fair amount of ancillary hardware that you have to budget for. Second, once you've bought a SAN you'll find that the initial cost is only the beginning. Most SAN vendors either require or strongly encourage you to purchase support and maintenance contracts. You'll probably want SAN management software, and you'll certainly need to make sure that your administrators are well-trained on SAN administration tasks—and that often means a smaller version of the SAN installed for test and training use.

Understand, too, that these additional costs are just for the basic stuff: hooking up hosts to the SAN so that they can share its disks. If you want to add capabilities such as geographically dispersed cluster nodes or hardware-based replication, expect the cost *and* knowledge requirements of your deployment to escalate significantly. As with clustering, the general recommendation for SANs for most organizations is that they shouldn't even be considered for deployment until the basic high-availability and disaster recovery measures described earlier in this book have already been taken.

### **Replication and Failover**

Replication itself is a useful way to safeguard the data on a server. However, by itself it doesn't provide service availability; once the data's been replicated, you still have to *do* something to make that data useful on the target server. Chapter 2 talked about replication as a disaster recovery technology. Most Exchange administrators see it in that light, but word is starting to get out: replication can be a terrific way to implement some advanced business continuance capability without jumping feet-first into a complete multi-site business continuance deployment.

The goals and design considerations for a business continuance-based replication deployment are similar to the ones discussed in Chapter 2 for disaster recovery-focused deployments: you have to ensure that there is sufficient bandwidth to replicate data between the source and target, and you must take measures to ensure that the replicated data are consistent. This latter requirement assumes extra importance for business continuance-driven replication. For disaster recovery, you can always fall back to tape if necessary, but in a situation that calls for your business continuance procedures to be activated, there may not *be* a tape backup available.

It's not necessary to rehash the discussion of how replication works. However, it's worth talking about how failover can be implemented. Windows cluster failover can be automatically triggered by a resource failure or manually triggered by an administrator; thus, it is with failover/replication solutions. The same basic set of resources needs to fail over: you need network identification data (a network name and IP address) so that clients can find the correct server after the failover, you need to update DNS records that point to the failed server so that mail can be delivered to it, you need the Exchange data itself (which is moved by replication), and you need a method to let clients and Active Directory (AD) peers (such as other Exchange servers) know that work has migrated over to the target server.

In general, the best way to make this happen is to update AD and DNS to indicate the presence of the new server, then update the clients so that they point to the new server, much as they would during a mailbox move. One of the primary differentiators between replication products for Exchange is how easy they make this process. Some products require you to write and run your own failover scripts; others can handle the server failover automatically, but require you to manually update client information to "notice" the new server. The best products can automatically update everything required for a client to find a given user's mailbox server. Updating the homeMDB attribute for each user whose mailbox was on the failed server will automatically point MAPI clients to the correct server when they connect, so that's the preferred solution. Outlook 2002 and later are smart enough to handle the sudden disappearance and reappearance of a failed Exchange server, whether you're using clustering or replication. With earlier Outlook versions, you'll probably have to quit and relaunch every client—but you'd have to do that with a Windows cluster solution, too.

## Design Choices

Once you make the decision to implement business continuance and high-availability features, the design aspects of your solution take on great importance because the amount of capacity you design in will directly affect the purchase *and* maintenance cost of your deployed solution.

### ***To Cluster, Or Not To Cluster?***

As I mentioned earlier, clustering offers some key advantages in the right circumstances. The trick to determining whether clustering makes sense for you is to dispassionately assess your environment to determine whether clustering is appropriate, or if you should be spending your money on more basic technologies first. The basic questions I recommend asking include:

- *What downtime causes are we trying to protect against?* Clustering helps protect against some specific single points of failure, and it makes rolling updates easier. However, there are other, more basic, single points of failure that you need protection from before you spend money on clustering.
- *What are my actual uptime requirements?* If you only need  $10 \times 5$  operations, there's not much point in paying for clustering's  $24 \times 7$  level of availability.

- *What is our organization's level of knowledge?* If, as an organization, you're not already comfortable administering Exchange and Windows, clustering is only going to bring you heartbreak.
- *Should I combine clustering with other technologies?* Many sites find that the one-two punch of clusters and SANs gives them superior flexibility in their disaster recovery and business continuance environments; others find that the ideal mix for their circumstances includes replication too.


For most organizations, the answers to these questions argue against using clustering. For them, the entry cost is too high for the benefits to be gained. If you have high uptime requirements that you can justify based on your *business* requirements, and if you have (or can get) the necessary skills to keep your cluster healthy and happy, it can be a very useful addition to your environment.

## SANs

The argument in favor of SAN technology is less ambiguous than for clustering, even though the two technologies share some common difficulties. Both are relatively expensive, and both require a good deal of specialized knowledge to use effectively (and safely!). Compared with clustering, though, the benefits of SANs are clearer. A properly designed SAN will give you:

- *Better performance.* Assuming that you put enough physical disks in your SAN and that you allocate them properly, SANs can put up some eye-popping performance numbers. Of course, you must be very careful when designing and implementing your SAN to follow good design principles (which Chapter 5 will talk about in more detail). For example, if you put your transaction logs and databases onto a single logical SAN volume, don't expect the same throughput that you would get if you separate them.
- *Multi-node cluster support.* Conventional 2-node clusters can use shared SCSI, and many do. However, for 3-node (or larger) clusters, SANs are preferable because they provide more stability and better performance.
- *More flexibility.* The ability to quickly add or reallocate storage without interrupting normal operations is a real blessing, as is the ability to move SAN volumes between servers. Depending on the kind of SAN you buy, you can choose between the vendor's hardware replication solution and host-based software replication, and most SANs now provide some degree of support for Microsoft's Volume Shadowcopy Service (VSS).
- *Better high-availability and business continuity capability.* SANs offer the promise of significantly better availability than DAS installations, along with support for technologies (such as VSS) that give you a better platform for building highly available Exchange solutions. Even without third-party products, SANs give you more options. To cite one example, if a server fails but your data's on the SAN, you can just move the data volume to another host and import it into a recovery storage group.
- *SAN booting.* One very useful SAN feature is being able to put *everything* the server needs—including the OS—onto SAN volumes. Thus, individual servers essentially become disposable; if one fails, just pop another one in its spot and let it boot from the SAN.

Of course, SANs have their drawbacks, chief among them being expense. A good RAID controller with a set of six 400GB drives gives you 1.2TB of RAID-1+0 storage—certainly adequate for most servers—for around \$3000. That same money wouldn't buy you very much Fibre Channel SAN hardware, although it would make a nice start on an iSCSI SAN enclosure. In addition, deploying a SAN requires you to perform due diligence beforehand to ensure that you have an adequate number of physical disks, suitably configured; the only real way to do so is with the Jetstress and Loadsim performance-testing tools from Microsoft, which means you have to commit to a certain degree of SAN deployment just to determine whether the planned deployment will support the desired numbers of users.

 Don't take a vendor's performance numbers at face value; there are too many ways to get incorrect performance data to make unaudited generic benchmarks trustworthy.

### **Failover and Failback Design**

Whether you're using clustering or a replication solution that provides failover/failback support, there are some design issues to consider *before* you put a solution into production. The idea of fully automatic, seamless failover is awfully attractive, but to get there, you need to do some preliminary work.

First, think of what you're going to fail over *to*. For replication solutions, one common mistake is to use an underpowered machine as the replication target just because it's available at the time. This setup usually works fine right up until that little uniprocessor server you tucked away in a remote location gets hammered flat by the sudden arrival of 2000 mailboxes' worth of user requests. Clusters have the same problem, but to a lesser extent; you can always add more cluster nodes. However, those nodes have to be appropriately sized. This area is one in which many sites got into trouble with Exchange 2000 clustering—they sized each node in the cluster to handle its own workload, so when a failover took place, the surviving node was suddenly staggering under double its normal workload. A better solution is to plan not to exceed 40 percent workloads: if you have two nodes, that means that neither node should run at more than 40 percent of its actual capacity so that the survivor will be able to handle the combined load, plus enough headroom for unexpected spikes in load. In clusters, you can fix this potential problem by either adding more nodes or by juggling the failover priority list.

Failover design is really more interesting from a non-clustering point of view because, depending on the solution you choose, there can be many more variables. Take a simple example: how do clients find the IP address of the server after a failover occurs? Normally, Outlook will use either DNS or WINS to find the IP address of the mailbox server specified in the user's profile. When the failover happens, clients need a way to get the updated IP address of the target server. To do so, you can change the IP address of the target server itself or update the DNS and WINS records for the server. These changes can be accomplished automatically, via scripts or tools that you start manually, or completely manually. Which is the best route? It depends on factors such as whether your remote server is attended (for example, is there someone there who can initiate failover if necessary?), how many DNS or WINS servers you have, replication delays attendant on any DNS updates that were necessary, and whether the group within your company that controls those servers will allow changes to be made when necessary.



There are other design considerations as well:

- *How is failover initiated?* Manual failover is obviously the most basic method, but automatic failover can shorten the interval that elapses before operations get back to normal after a failure. However, you might not want to automatically fail over normal operations so that you don't accidentally cause a failover during transient problem conditions.
- *How hard is it to fail back to the original node after a failover?* A solution that offers easy failover but only complicated failback is probably not going to be very useful for anything other than catastrophic failures.
- *How often can you replicate?* Depending on how much bandwidth you have, and what kind of replication solution you use, you may not be able to replicate all that often. For example, let's say that you replicate once every 4 hours—in between those 4-hour replication updates, a failure will cause data loss. That's acceptable *if your SLA allows it*, but you should be aware of the possibility as part of your planning.
- *How long does failover actually take?* If your failover solution uses replication, there will probably be an interval between the start of a failover operation and its completion, just as there is with clustering; you obviously want to keep this interval as short as possible. It's equally important to assess how long failback takes, particularly if failback can't be completed until you've replicated changes from the failover server back to the original source server.
- *How seamless are failover and failback operations to the client?* Windows clustering can make the failover completely transparent to Outlook 2002 and later because they're smart enough to retry failed connection requests. However, failover solutions that require updates to user objects in AD will probably require users to quit Outlook and relaunch it (and they may even require users to log off and back on again). This is no big deal as long as your users know about it ahead of time.

### **Planned vs. Unplanned**

As mentioned in preceding chapters, it's worth examining how the technologies described in this chapter can help with both planned and unplanned downtime. After all, the whole idea behind high availability is to minimize unplanned downtime while simultaneously helping to shorten the amount of time required for planned maintenance. How do these technologies measure up?

First, clustering. Clearly, properly implemented clusters can help reduce the amount of unplanned downtime by providing automatic failover; the less obvious, but perhaps more important, benefit they offer is their ability to make planned maintenance much simpler by hiding it from users. For example, let's say that you want to install an Exchange service pack. Without clusters, you need to back up each server, stop its Exchange services, install the service pack, and restart the Exchange services. Of course, during this process, you might find that you have to fix problems that crop up. With clustering, the process is much simpler: you fail one node's resources over to another, install the service pack on that node, then fail the resources back. (You probably still want to take a back up!) This process helps transform Exchange and Windows service pack installation from work that must be done during periods of low user demand to work that can be accomplished when it's most convenient for your staff.

The value of this benefit has declined somewhat as Microsoft has gotten better about not requiring reboots for many security fixes, but bear in mind that you also need to use planned maintenance time to install updates to other software on your Exchange server, such as antivirus scanners, replication software, and backup tools. These add-ons may require patches, reboots, or other unwelcome intrusions that clustering can help mitigate.

SANs don't offer much help with planned downtime; if anything, they require *more* maintenance than DAS solutions. However a well-built SAN can make a huge difference in your unplanned downtime numbers by insulating you against many types of storage subsystem failure. Of course, SAN technology isn't a silver bullet that magically protects against *everything*, but it can definitely help.

Replication and failover technologies have a role to play here too. Their application to unplanned downtime is obvious; for planned downtime periods, the ability to fail over work to a remote node gives you many of the benefits of a cluster without the burden of actually implementing and maintaining one. This benefit is quite attractive.

## Summary

Planning and implementing capability for high availability and business continuance is quite a bit different from planning pure disaster recovery capabilities. Clustering, SANs, and replication and failover technologies all have their uses, and understanding which options to implement where is an important part of your overall system design.

## 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.realtimedpublishers.com/contentcentral/>.