

realtimepublishers.comtm

The Definitive Guidetm To

Scaling Out SQL Server 2005

Don Jones

Chapter 6: Windows Clustering.....120

Clustering Overview120

 Clustering Terminology120

 How Clusters Work.....121

 Cluster Startup122

 Cluster Operations123

 Cluster Failover.....124

 Active-Active Clusters.....126

Clusters for High Availability.....128

Clusters for Scaling Out.....128

Setting Up Clusters132

SQL Server and Windows Clusters137

Clustering Best Practices138

Optimizing SQL Server Cluster Performance140

Case Study140

Database Mirroring142

Summary.....143

Copyright Statement

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

Realtimerepublishers.com and the Realtimerepublishers 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 Realtimerepublishers.com, please contact us via e-mail at info@realtimerepublishers.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 6: Windows Clustering

You're considering a scale-out solution to ensure that the many users who need access to data can do so without worrying about downtime. In fact, your organization wants to avoid downtime at all costs. Enter Windows clustering. Although Windows clustering isn't a requirement in a scale-out solution, it offers a higher level of availability than standalone servers can provide.

 I want to emphasize that Windows Clustering isn't a scale-out solution in and of itself; it is, however, a common addition to scale-out solutions because it provides the high availability that scale-out solutions often require. SQL Server 2005 also provides database mirroring, a high-availability solution that provides similar capabilities. In addition to Windows Clustering, this chapter will briefly discuss database mirroring.

Clustering Overview

Microsoft has offered clustering as an option since NT 4.0, Enterprise Edition. In Win2K, only the Advanced Server and Datacenter Server editions include clustering capabilities; with WS2K3, the Standard Edition also includes the clustering capability.

 There are several non-Microsoft solutions for clustering SQL Server, many of which also provide SQL Server-specific advantages such as real-time replication capabilities. However, for this chapter we'll focus on the Windows Cluster Service software provided with Windows.

Clustering Terminology

Before we explore clustering in more detail, it is important to define some basic terminology to prevent confusion. The following list highlights the essential clustering terms:

- **Node**—A single server within a cluster. It's called a node to distinguish it from other non-clustered servers.
- **Cluster**—Any collection of one or more nodes. Even a cluster with just one node is considered a cluster: If you have a 2-node cluster, and one node fails, you're left with a 1-node cluster.
- **Virtual server**—End users and client applications don't connect directly to cluster nodes; they connect to virtual servers, which represent specific services—such as file sharing, Exchange Server, and SQL Server—that the cluster can provide. Virtual servers can be passed back and forth across cluster nodes, allowing the service to remain available even if a particular node isn't.

- Small Computer Systems Interface (SCSI)—One of the most popular means of connecting high-speed storage, normally in the form of hard drives, to a computer. Although SCSI is most often thought of in terms of copper cabling, it's also used in Fiber Channel implementations, iSCSI implementations (running SCSI over normal Ethernet connections), and so forth. SCSI is an important part of Windows clustering.
- Failover—When a cluster node fails, failover is the process used by other nodes in the cluster to assume the responsibilities of the failed node. An administrator can perform a manual failover, transferring services from one node to another in order to take a node offline for maintenance.

How Clusters Work

Windows Clustering is based on a *shared-nothing* model, which means that none of the cluster nodes have access to the same resources at the same time. The easiest way to examine Windows clustering is to consider a simple, two-node, active-passive cluster running a single instance of SQL Server, such as the one that Figure 6.1 shows.

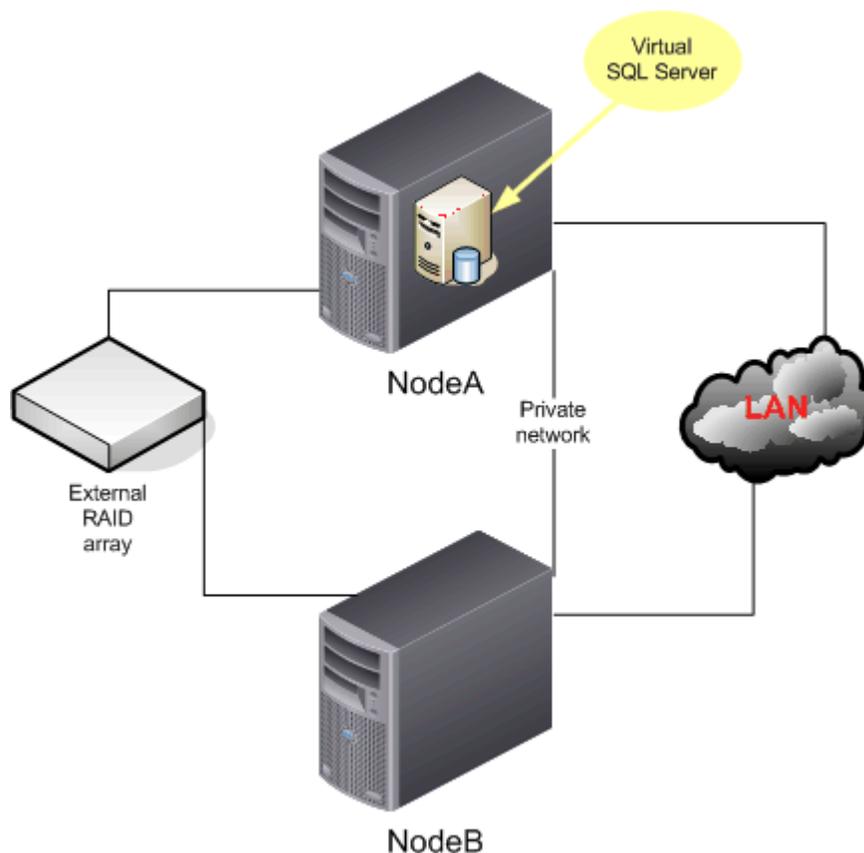


Figure 6.1: Basic single-node, active-passive cluster.

In this example, two servers are nodes in the cluster. Each node provides private storage for the OS and any applications the cluster will run, such as SQL Server. This private storage can be in the form you prefer, such as internal hard drives or an external drive array.

The nodes are also connected to a single external drive array. *All* of the nodes are connected to this array, but they can't all access the array at the same time. The external array can be as simple as a RAID cabinet provided by your server vendor or more powerful, such as an EMC storage cabinet. Regardless, the external storage must be configured to have at least two logical volumes: one large volume will be used to store data from clustered applications such as SQL Server, and a small volume is required for the cluster's *quorum resource*, a file that describes the cluster's configuration.

 The "shared" external drive array provides a single SCSI bus. Both nodes connect to this bus, although their controllers must have different SCSI device ID numbers. Only one computer can successfully communicate over the bus at a time; thus, the Windows Cluster Service controls the nodes' communications over the bus. As a result of the special level of control required, only certain SCSI array controllers that provide cluster-compatible drivers can be used in a cluster configuration.

Also note that the nodes share a network connection to one another. This connection can be as simple as a crossover cable, or the connection can be run through a more traditional hub or switch. This private network connection will carry the *heartbeat* signal, a continuous pulse that proves the cluster's active node is still functioning. You *could* run this heartbeat over the regular network connection that connects the nodes to the rest of your network, but you run the risk of occasional spikes in network traffic delaying the heartbeat. A delayed heartbeat could result in unnecessary failovers; thus, it is best to use a dedicated connection.

Cluster Startup

When you start up the first node in the cluster, it runs Windows normally. When the Windows Cluster Service starts, the service performs a bus reset on the shared SCSI array. After performing the reset, the service pauses to determine whether another attached node performs a similar reset. When none does (because no other node is turned on yet), the node determines that it is the first node in the cluster and immediately begins starting all clustered resources.

Clustered resources typically include the external storage array, one or more virtual computer names, one or more virtual IP addresses, and clustered applications such as DHCP, Exchange Server, and SQL Server. Clustered applications' executable files are stored on the node's private storage; their data is stored on the cluster's shared external array. Clients use the virtual computer names and IP addresses to talk to the cluster and the clustered applications; because any node in the cluster can potentially respond to these virtual names and addresses, clients will always be able to contact the cluster even if a particular node isn't available.

When the second (and subsequent) nodes are started up, they also perform a SCSI bus reset. However, the first node owns the external storage resource at this time, so it immediately performs its own bus reset. The second node sees this reset, determines that the cluster is already running, and assumes a passive role. In this role, the second node simply monitors the incoming heartbeat signal and waits for the active node to fail. Any clustered services—such as SQL Server—are held in a suspended state rather than started normally. Figure 6.2 illustrates the cluster's condition.

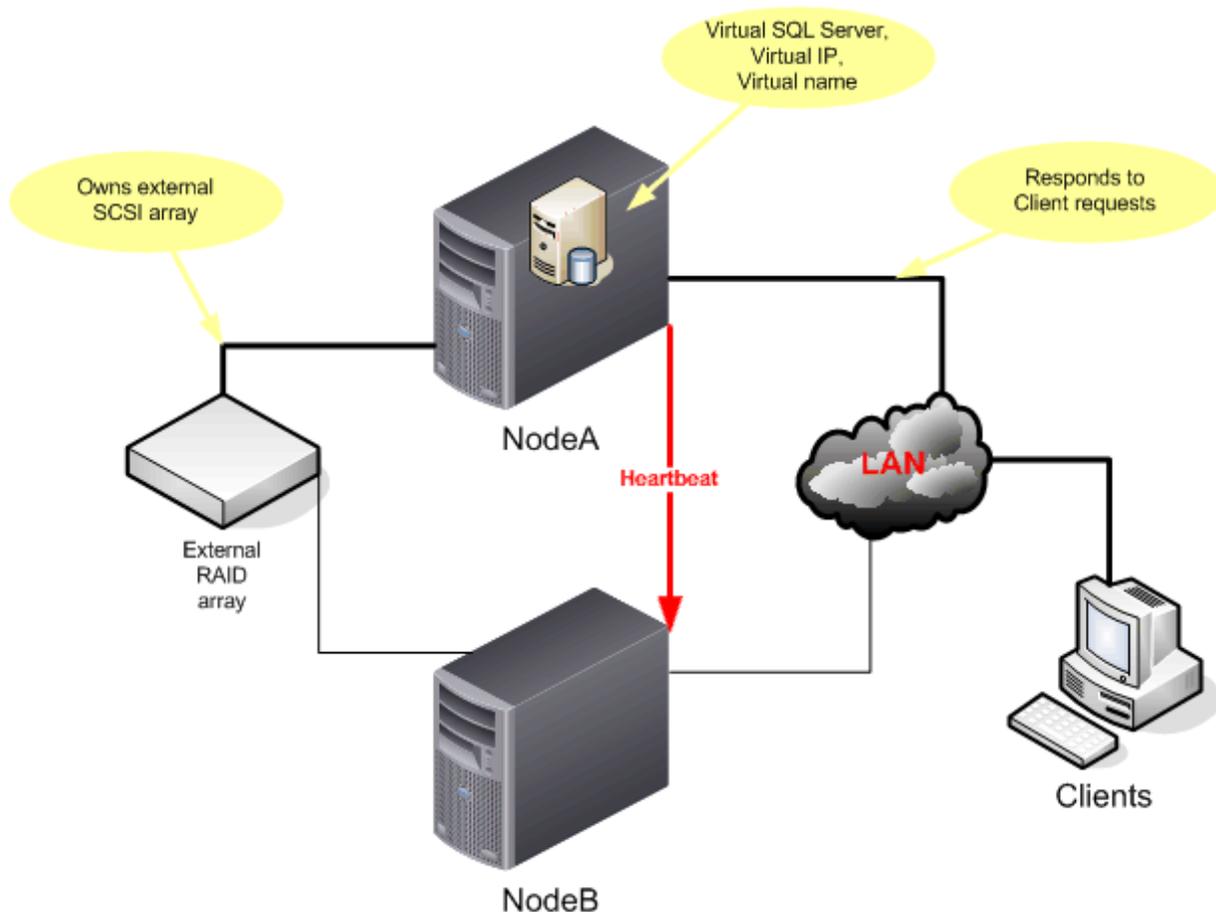


Figure 6.2: An active and a passive node in a 2-node cluster.

The passive node is only passive with regard to the cluster. In other words, it's possible for the passive node to perform useful work because it is a fully fledged Windows server. The node simply focuses on non-clustered applications. For example, you can run a reporting application on the passive "spare" node, allowing the node to be useful while acting as a backup for the functioning active node.

Cluster Operations

While the cluster is working, incoming traffic is sent to the cluster's virtual IP addresses. The active node responds to this traffic, routing it to the appropriate applications. In fact, the only practical difference between the active node and any other Windows server is that the node is sending a heartbeat signal to the other nodes in the cluster.

Interestingly, the clusters don't use a virtual MAC address to respond to incoming traffic. When clients (or a router) needs to forward traffic to the active node, the Address Resolution Protocol (ARP) sends out a request for the MAC address, including the requested cluster IP address in the request. The active node sees the request and responds with its own MAC address. Should the active node fail, a few requests might be sent before clients (and routers) realize that they are not getting a response; in that case, the client (and routers) would resend the ARP request, and whichever node had taken over for the failed node will respond with its own MAC address.

Thus, clustered applications' client components must be willing to resend requests in order to re-establish connectivity when a cluster node fails and the passive node takes over. One reason that SQL Server works so well as a clustered application is that Microsoft wrote both the client and server end: Although you might have a custom client application running on your users' computers, that application is probably using Microsoft's ActiveX Data Objects (ADO), Open Database Connectivity (ODBC), or ADO.NET in order to connect to SQL Server. Those database connectivity objects will automatically resend requests to the server as needed, instantly making your custom client applications cluster-aware.

While the cluster is running, you can transfer the cluster's active node responsibility from node to node. Although similar to a failover, this process is much more controlled. Essentially, you transfer a group of cluster resources—such as a virtual computer name, IP address, and SQL Server service—from one node to another. On the newly active node, those services will begin to start, while at the same time they begin to shut down on the now-passive node. Generally, a transfer of resources from one node to another takes about half a minute or less, depending upon the specific applications and services involved. Transferring services in this fashion allows you to perform maintenance on cluster nodes while keeping the overall clustered application available to your users.

Cluster Failover

At some point, an active cluster node will fail. When it does, its heartbeat signal stops, telling the passive node that there is a problem. The passive node performs a bus reset on the external, shared SCSI bus array. When the formerly active node doesn't perform its own reset, the passive node determines that the other node has failed.

 The SCSI bus reset step is an extra precaution. If the heartbeat signal had failed momentarily due to a network problem, the SCSI bus reset would keep the passive node from seizing control of the cluster when the active node is still working. When both steps—the heartbeat and the bus reset—fail, the passive node knows it's time to step in and take over.

The passive node now seizes control of the cluster, appointing itself active node. It quickly reads the quorum resource to determine how the cluster is currently configured, and begins starting clustered services and applications. It also begins responding to the cluster's virtual IP addresses and names. Within about 30 seconds, the passive node is the active node. Figure 6.3 illustrates failover in a sample cluster.

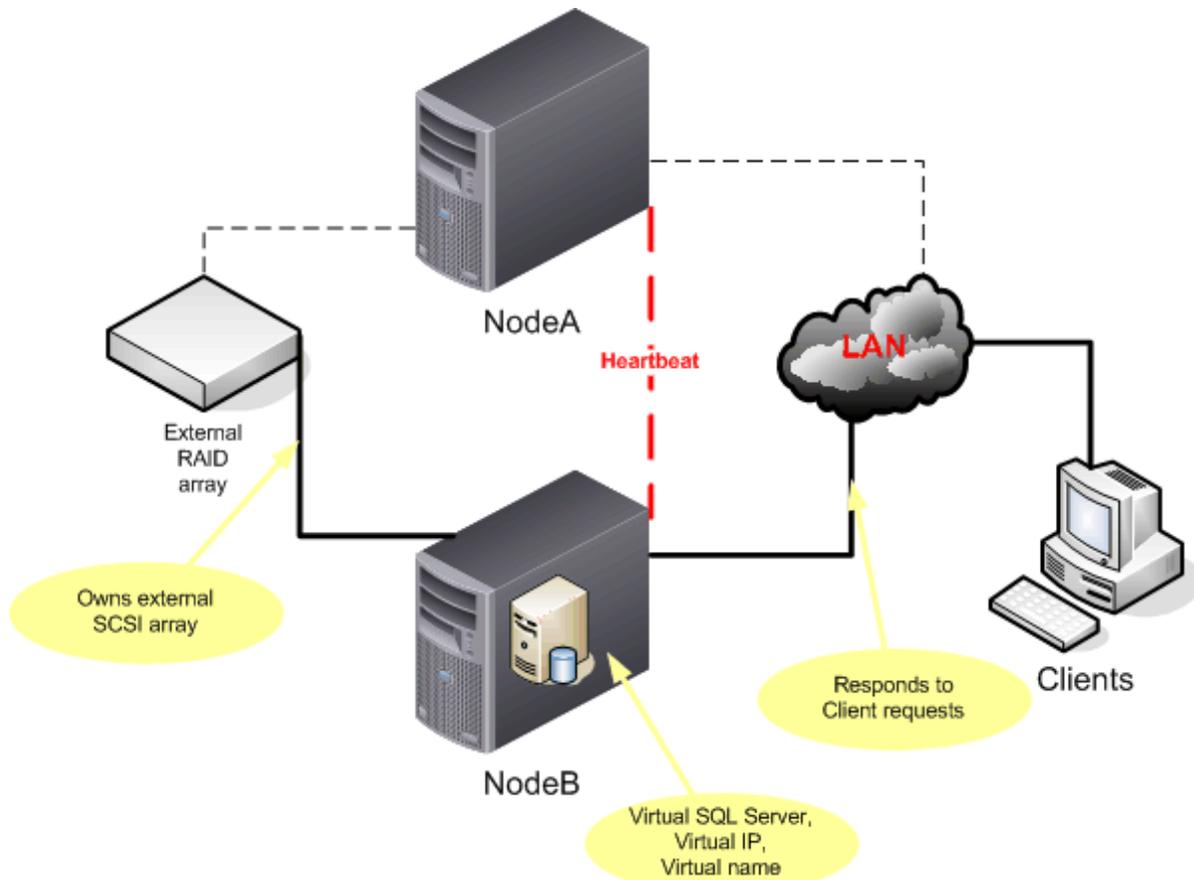


Figure 6.3: Failover of cluster resources to the passive node.

Windows Clustering supports a concept called *failback*, where the cluster will attempt to shift clustered resources back to the original, preferred node. For example, suppose you built your cluster so that the passive node is performing other, non-clustered work, and you don't want it being the active node in the cluster for any longer than necessary. To configure this preference, you designate within the Cluster Service that one node is the *preferred* node for clustered resources. Whenever that node is online, all resources will be transferred to it. If it fails and is subsequently restarted, all cluster services will transfer back to it once it is online again.

However, the back-and-forth of clustered resources across nodes can prove annoying and disruptive to users. To prevent disruption, you can configure failback policy to only occur during evening hours, and to stop occurring if the preferred node fails a certain number of times.

Active-Active Clusters

To eliminate the waste of having one server sitting around in case the other fails, you have the option to create *active-active* clusters. In an active-active cluster, each node performs useful work and is backed up by the other nodes. For example, consider the cluster in Figure 6.4.

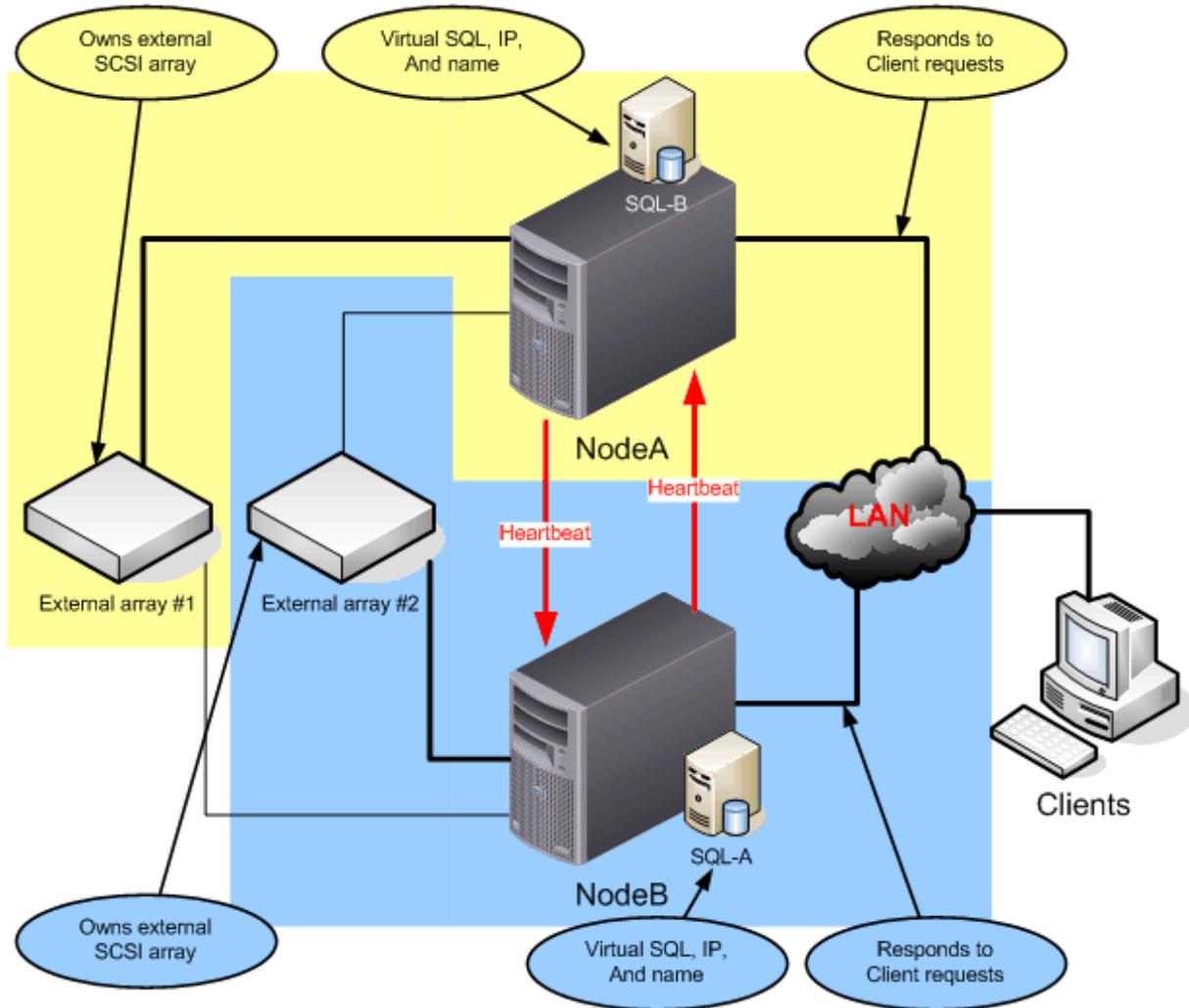


Figure 6.4: An active-active, 2-node cluster.

This figure illustrates two logical clusters implemented across two nodes. Each logical cluster has a set of resources, including an external drive array that is connected to both nodes, a virtual IP address, a virtual name, and one or more clustered applications. Under normal circumstances, each node owns one set of resources, just as a standalone server might. One cluster is highlighted in yellow, and the other in blue.

When a failure occurs, one node can own *both* sets of clustered resources, effectively becoming two servers on one machine. Depending on how you design the cluster, its performance might be much lower than it was when both nodes were running; but *poor* performance is often better than *no* performance. Figure 6.5 shows the failed-over configuration, with one server handling both logical clusters.

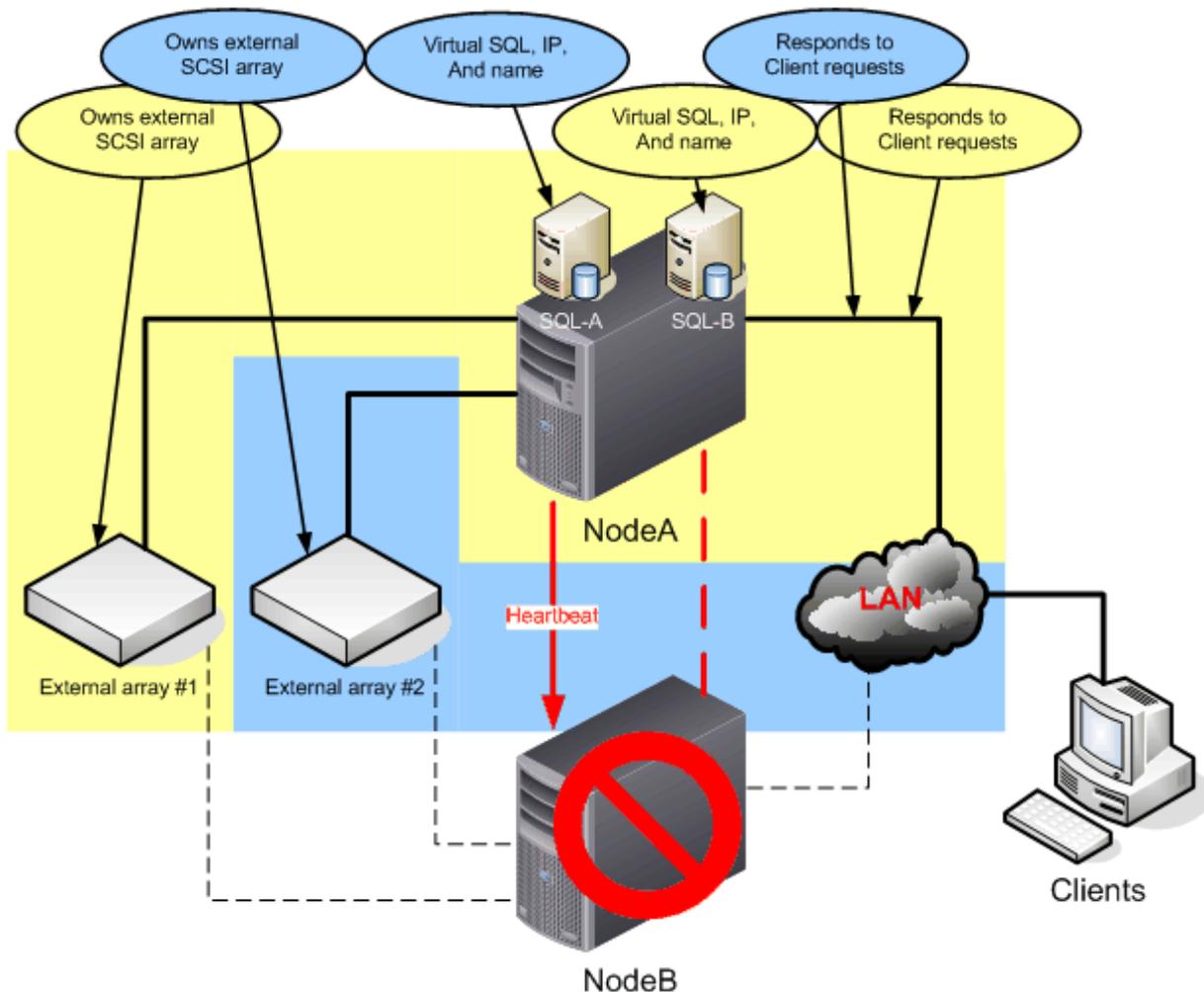


Figure 6.5: An active-active cluster in failover condition.

Windows Clustering in later editions of Windows, such as WS2K3 Enterprise Edition and Datacenter Edition, can handle more than just two nodes in a cluster. In fact, it's not uncommon to have 4-node clusters running SQL Server in an active-active-active-active configuration. In these configurations, each node is a functioning SQL Server, and any node can take over for the failure of any other node. In fact, it's theoretically possible for three of the four servers to fail and for all four logical clusters to continue serving clients. However, to actually achieve this level of redundancy, you would need to engineer each cluster to handle no more than 20 to 30 percent of its maximum capacity under normal conditions. That level of over-engineering can be expensive, which is why most cluster designers target 60 percent utilization, allowing each node to carry the load of two nodes with only slightly degraded overall application performance.

Clusters for High Availability

Clustering obviously offers advantages when building highly available solutions of almost any kind. However, before you assume that clustering is the perfect solution to high availability, consider one important fact: Clustering only provides protection against total hardware failures in either node. It does *not* protect against a hardware failure in the external drive arrays, nor does it protect against software failure that corrupts application data.

External drive arrays are usually built in RAID arrays, offering a level of protection against a single drive failure. Nothing, however, can protect against a corrupted database file. For example, suppose you build a 2-node, active-passive SQL Server cluster. A year later, one of your MDF database files is damaged. No amount of clustering will help, because clustering doesn't provide redundancy for that database file. You will still need to have a good backup in order to restore the database, and your database is going to be unavailable until you do restore it.

Database Mirroring in SQL Server 2005

One weak point in Windows Clustering is the data store. As I've described, each virtual SQL Server instance has only one copy of its data, which is stored on some form of external storage, such as a Storage Area Network (SAN) or a directly connected external storage array. If that data store becomes corrupted, the virtual SQL Server is useless. In other words, Windows Clustering provides high availability only for the cluster's *hardware*, not its data.

SQL Server 2005 introduces a new technology called *database mirroring*. Essentially, you maintain a completely independent, spare SQL Server 2005 computer, that *mirrors*, or copies, one or more databases from one or more other SQL Server 2005 computers. These mirrors are continuously updated, so that if the primary server fails, the backup server can be pressed into service with a minimum of downtime. SQL Server 2005 provides techniques to help that "failover" occur quickly and automatically.

Database mirroring is, in fact, a kind of clustering (using the term *clustering* in a generic sense), although the servers in the "cluster" don't share the same close relationship as those in a Windows cluster do. Because mirroring is configured on a *per-database*, rather than *per-server*, basis you have a lot of configuration flexibility: A single backup server can contain mirrors, for example, from many production servers.

Like Windows Clustering, database mirroring doesn't provide load balancing, meaning it isn't, in and of itself, a scale-out solution. It does, however, provide the high availability that many scale-out solutions require.

Clusters for Scaling Out

In Chapter 4, I showed you how to create scale-out solutions using distributed, partitioned databases. In these databases, the actual database data is partitioned horizontally and spread across two or more servers, which are referred to as a *federation* of servers. This federation works together through distributed partitioned views to act as a single, giant SQL Server: Each federation member contributes a portion of the data necessary to fulfill queries, allowing various pieces of the query to be executed in parallel for much faster results. Distributed partitioned views are a good scale-out technique in certain situations. From a software development standpoint, they are one of the easiest scale-out techniques to implement on SQL Server; however, they don't offer the best performance in every situation. Figure 6.6 provides a simplified illustration of how a distributed partitioned view works with a server federation.

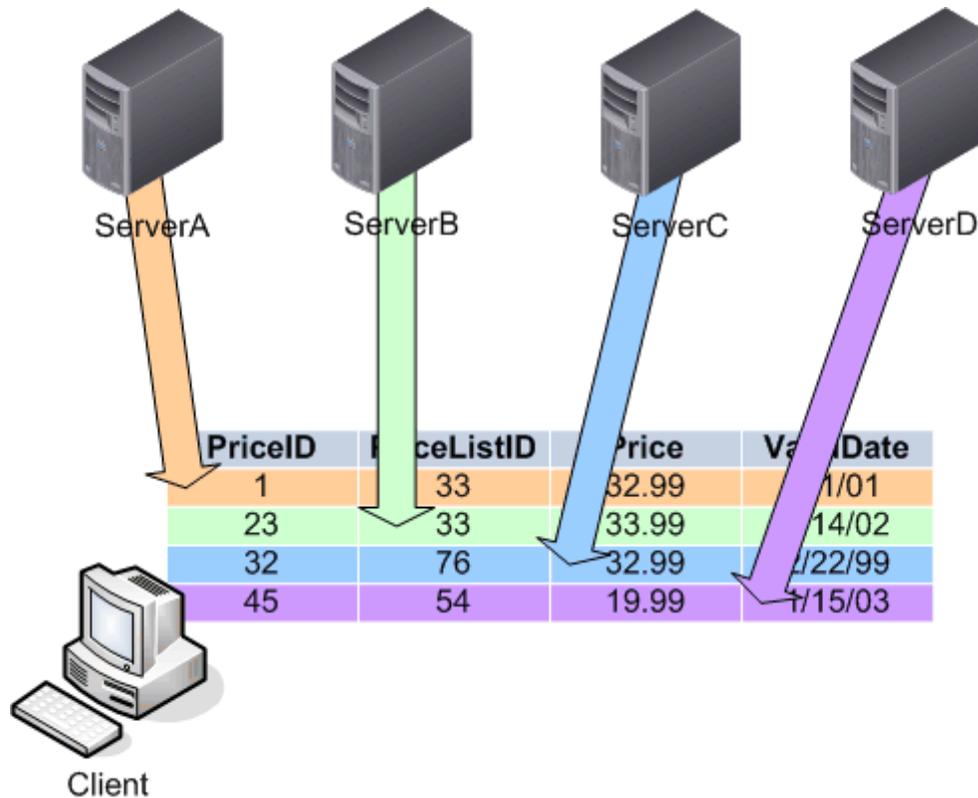


Figure 6.6: Distributed partitioned views allow multiple servers to contribute to a single set of query results.

Another technique is *data-dependent routing*. In this technique, a custom middle tier of your application takes the place of the distributed partitioned view, handling the routing of queries to the server or servers that contain the required data. This technique is useful in cases in which the data can't be horizontally partitioned in such a way to have most users querying most of their data directly from the server containing that data. In other words, when a distributed partitioned view would have to acquire a significant amount of data from another server on a regular basis, data-dependent routing provides better performance. However, data-dependent routing requires significantly more development effort and might less-readily accommodate back-end repartitioning.

Distributed partitioned views and federated servers, however, are extremely vulnerable to hardware failure. A single hardware failure in any of the four servers that the figure shows could result in the entire application becoming unusable because one-fourth of the application's data is unavailable. The result is that a single, even minor, failure—such as a processor power module or server power supply failure—could result in all four servers, and the entire application, being completely useless.

Clustering can help. By implementing the four servers in an active-active-active-active cluster (or even as two independent active-active clusters), the failure of a single piece of hardware won't affect the availability of the overall application. In this case, clustering isn't providing a scale-out solution by itself, but it is contributing to the reliability of an existing scale-out solution. Figure 6.7 shows a more detailed view of how the federated servers might be built into a 4-node cluster.

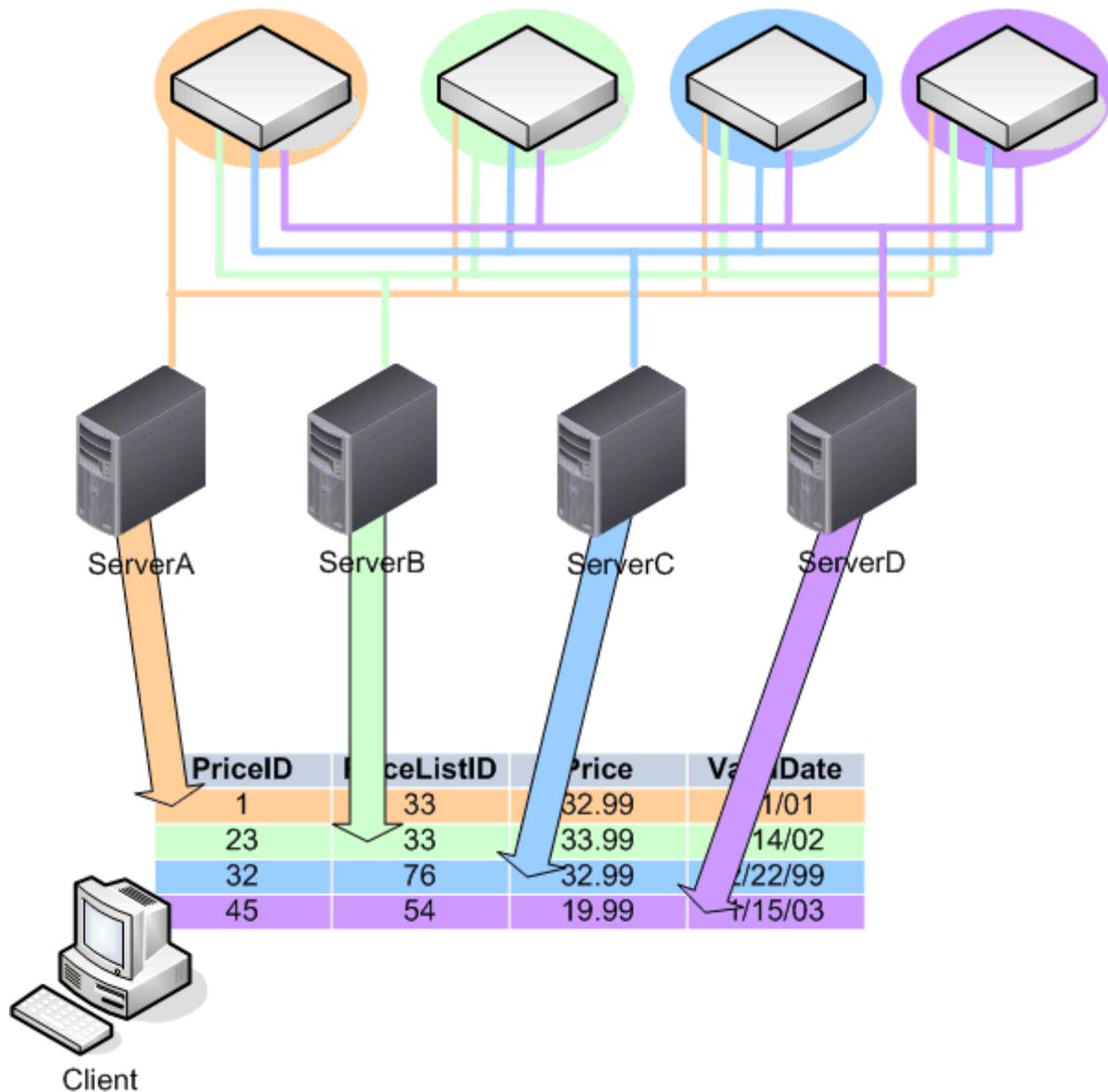


Figure 6.7: A 4-node cluster in a server federation.

In the event that a single server fails, one of the others can take over for it, acting as two virtual SQL Server computers while the one node is offline. Figure 6.8 illustrates how the cluster failover process ensures that all of the application's data is available, even when a single federation member is unavailable.

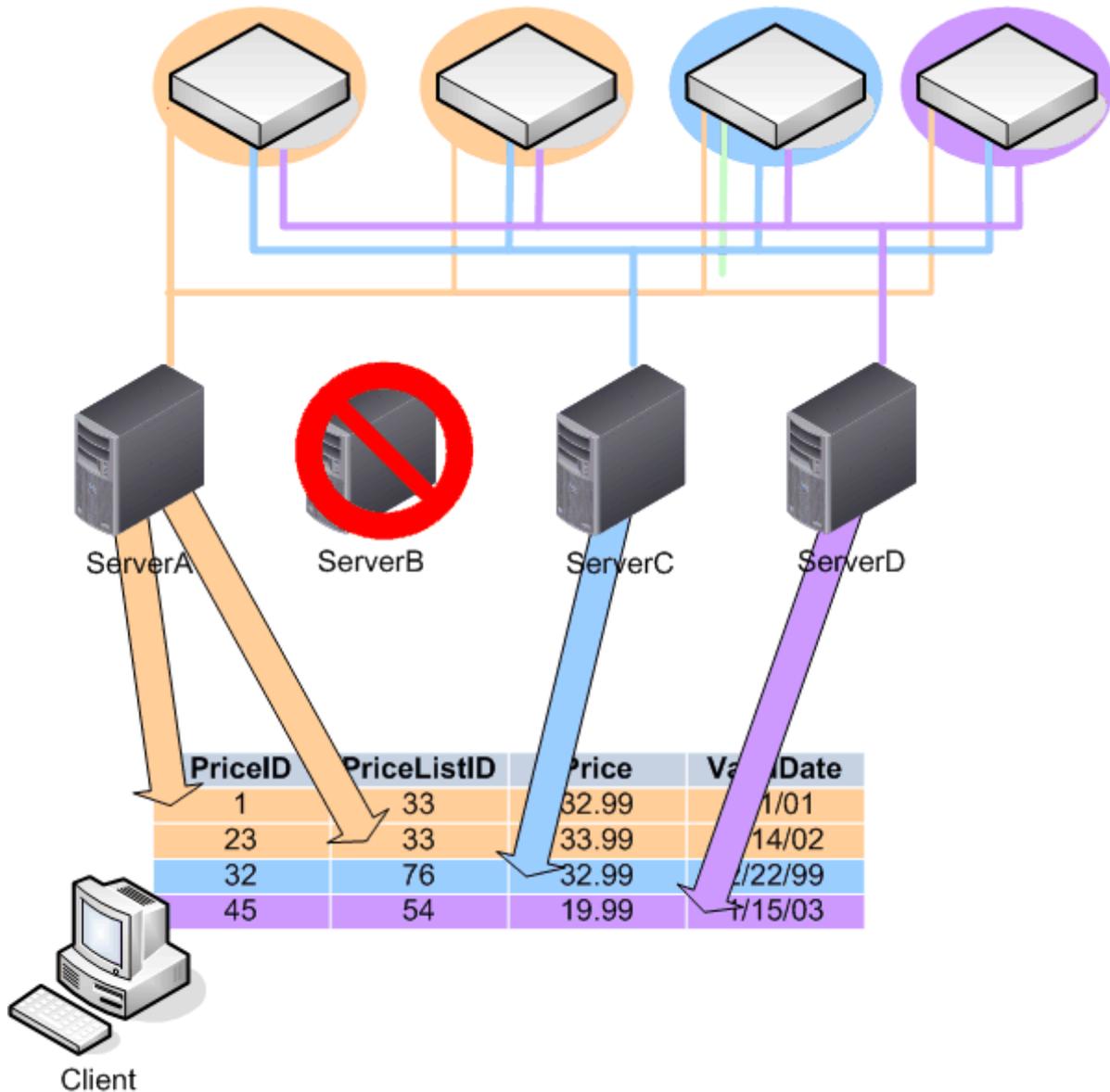


Figure 6.8: Clustering provides fault tolerance in a server federation.

The distributed partitioned views will probably run somewhat slower when one server must carry the workload of two, but slower performance is usually more acceptable than the entire application simply being unavailable due to a single server failure.

Setting Up Clusters

Setting up a cluster is a fairly straightforward process. You can perform the setup completely remotely using WS2K3's Cluster Administrator console. Simply launch the console, and when prompted, select the action to create a new cluster, as Figure 6.9 shows.



Figure 6.9: Creating a new cluster.

Next, you'll provide some basic information about the new cluster, including its domain. Cluster nodes *must* belong to a Windows domain so that the clusters can communicate by using a single user account. Without a domain, it's impossible for two servers to share a user account. Figure 6.10 shows the dialog box in which you'll enter the domain information and the proposed name of the new cluster.

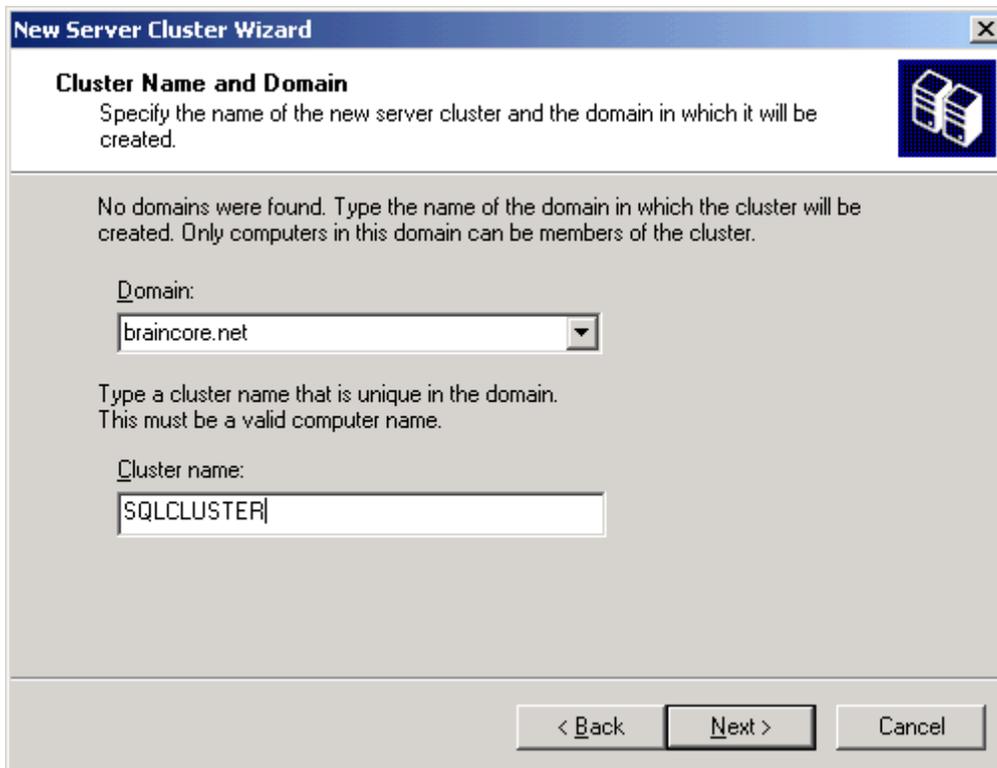


Figure 6.10: Entering the cluster name and domain.

Next, you'll provide the name of the first node in the cluster. This node must be an existing server, and it must meet the pre-requisites for being a cluster node. Figure 6.11 shows the dialog box in which you enter this information.



Figure 6.11: Selecting the first cluster node.

Next, the wizard will attempt to verify the information you've entered and determine whether a cluster can be created. A status dialog box, which Figure 6.12 shows, keeps you apprised of the wizard's status.

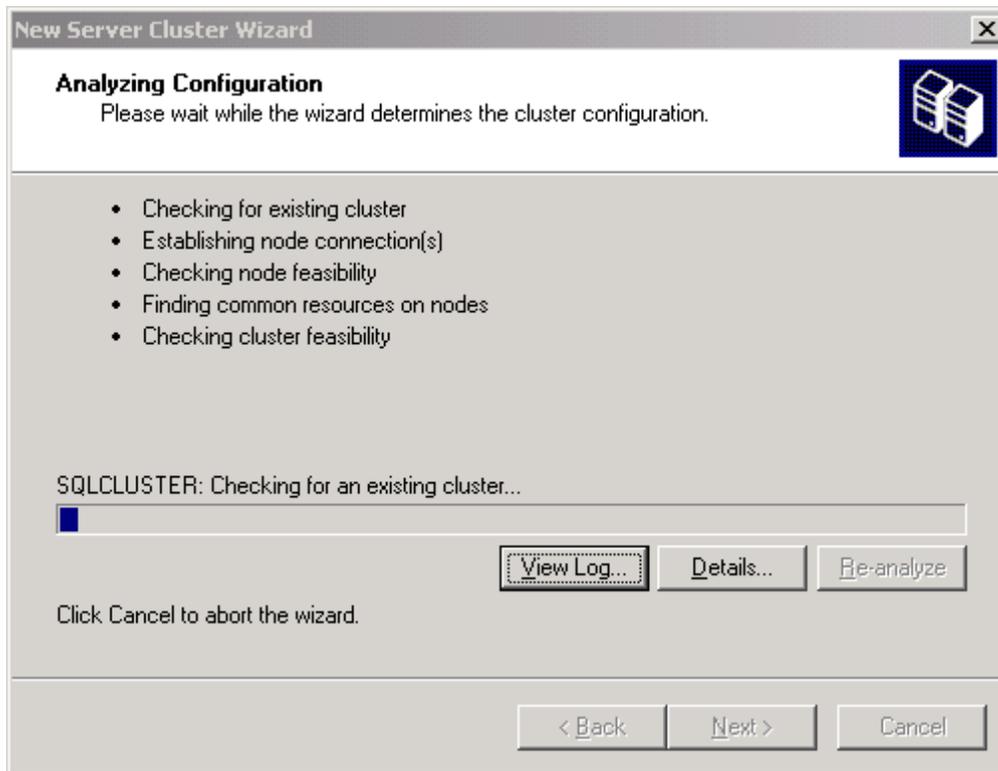


Figure 6.12: Analyzing the proposed cluster configuration.

Figure 6.13 shows the dialog box that the wizard presents when the node you specified isn't suitable to be a cluster node. This dialog box is common because Windows is picky about cluster requirements.

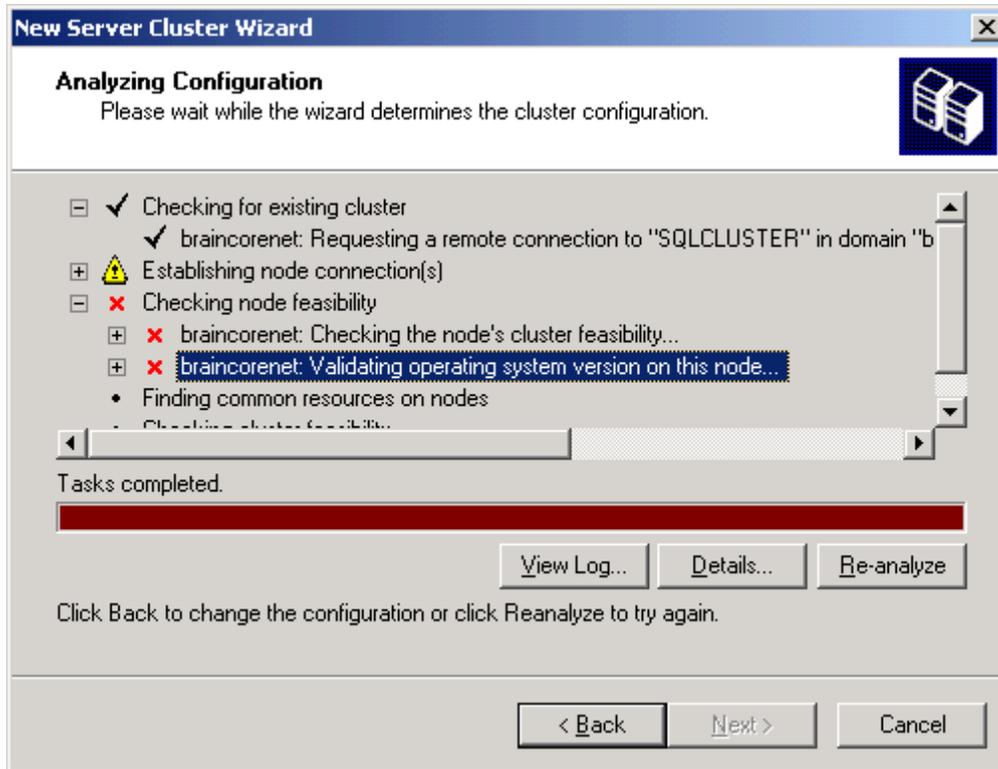


Figure 6.13: Cluster pre-creation check failed.

The following list provides the pre-requisites for a Windows cluster:

- The nodes must all belong to the same domain.
- You must run the cluster creation wizard as a member of the Domain Admins group in the domain that contains the cluster nodes.
- The first cluster node must have internal storage and at least two other logical volumes. These volumes will be used to contain the cluster quorum resource and application data, respectively. These volumes must also be on a SCSI controller—not an IDE controller.
- Each node must have the correct OS—either Enterprise Server Edition or Datacenter Server Edition for cluster nodes.
- Disks shared between the nodes must be basic disks; they cannot be dynamic disks or spanned volumes. The disks must be formatted with NTFS.



Some third-party products allow for the use of dynamic disks in a cluster; search the Microsoft Knowledge Base for “server cluster dynamic disks” for updated information.

- Each node must have a static IP address, and the cluster itself must have an additional static IP address. You will need yet another static IP address for each virtual SQL Server instance you create.

Assuming your nodes meet the requirements, the cluster creation wizard will complete successfully. You will run the wizard again to add the second and subsequent nodes to the cluster, and when you're finished, you'll have a complete cluster containing however many nodes you specified.

When completed, your cluster will contain basic resources. These resources are initially assigned to the first node in the cluster, although you can transfer them as you like. The resources are organized into *resource groups*, and, generally speaking, all of the resources in a resource group are dependent upon one another and must be transferred as a group.

For example, one of the most important resources is the Cluster Disk resource, which represents the cluster's external disk. The quorum resource, which represents the cluster's quorum configuration, resides on the Cluster Disk, and so must be transferred with it. Likewise, the cluster's virtual name and IP address also depend on the Cluster Disk and must be transferred with it. In an active-active cluster, you'll have *multiple* cluster disks, names, and IP addresses, which can be transferred, as a group, independent of the other groups, as the following example illustrates.

Resource Group A	Resource Group B
Cluster Disk E: Quorum Resource Cluster name "ClusterA" Cluster IP "192.168.0.25"	Cluster Disk F: Quorum Resource Cluster name "ClusterB" Cluster IP "192.168.0.26"
Owned by Node A. Potential owners: Node A, Node B	Owned by Node B. Potential owners: Node A, Node B

In this active-active configuration, either node can own either resource group or one node can own both resource groups if the other node happens to be offline. SQL Server is installed as another set of resources, which are dependent upon cluster disks, IP addresses, quorums, and so forth.



You can create multiple quorums, IP addresses, and other shared resources within a cluster. Suppose, for example, that you have a cluster running both SQL Server and Exchange Server. The cluster contains two nodes. Node A contains an instance of SQL Server and an instance of Exchange Server. Node B contains an active instance of SQL Server.

With the right combination of disks, quorums, and other resources, you could transfer Exchange Server, for example, to Node B independently of Node A's SQL Server instance. Or, if Node A fails, both its SQL Server instance and Exchange Server instance could be transferred to Node B—which would then be running Exchange Server and two copies of SQL Server. Such a configuration is not recommended, but is useful to illustrate the cluster's capabilities.

SQL Server and Windows Clusters

SQL Server is easy to set up on a cluster. First, create the cluster, including all the nodes. Ensure that all the nodes are turned on, and test the cluster's failover capabilities with the basic resources created by the cluster creation wizard. Next, simply install SQL Server on one of the cluster nodes.

SQL Server will automatically detect the presence of Windows Clustering on the node and install in a clustered configuration. Each node will receive the proper SQL Server software, and the necessary clustered resources will be created in the Cluster Administrator console. You can then use Cluster Administrator to configure a preferred node for SQL Server, if desired, and to configure other cluster-specific settings.

During installation, you'll specify a virtual server name and an IP address for SQL Server to use. Clients will use this name and IP address to access the SQL Server databases on the cluster. If you're creating an active-active cluster, simply repeat SQL Server Setup for each active node. In each instance, specify a unique SQL Server virtual server name and IP address. Each time you run SQL Server Setup in this fashion, you'll create a new active instance of SQL Server on the cluster, including the proper cluster resources in Cluster Administrator.

For example, in a 2-node, active-active cluster, each node will contain two separate instances of SQL Server, which you'll be able to see in the Services console. SQL Server utilizes its multi-instance capability so that each cluster node runs two distinct instances of SQL Server's services, including SQL Server Agent and the Distributed Transaction Coordinator (DTC). The services are only started on the active node: Instance 1 might be started on Node A, while Instance 2 is started on Node B. In the event Node B fails, then Node A will start its copies of the Instance 2 services, allowing Node A to effectively act as two SQL Server computers.



This behavior isn't unusual. Remember that you can install multiple named instances of SQL Server on any Windows computer, allowing that computer to respond as if it were multiple SQL Server computers. Clustering simply coordinates the failover and service startup between multiple computers.

For example, it's possible—although it sounds complicated—to have a 2-node cluster with four instances of SQL Server. Node A might normally run Instances 1 and 2; Node B would normally run Instances 3 and 4. You would run SQL Server Setup four times to create this configuration, and if either node failed, Windows Clustering would move all available instances to the surviving node.

Clustering Best Practices

There are a few practices related to clustering that are a good idea to follow. These best practices will help you avoid problems, maximize performance, and reduce frustration:

- Configure all cluster nodes to meet the clustering requirements before beginning. This step includes formatting partitions on both the nodes' private storage as well as on the external storage that the nodes will share.
- Use identical network adapters in all nodes (down to the firmware version). Use built-in network adapters only if there are enough to meet the cluster's needs (usually two).
- Provide separate network connections for the private heartbeat connection and public client connections.
- Avoid using teaming network adapters on the private heartbeat connection; they can cause problems that make the heartbeat appear to fail.
- Bind network adapter protocols so that the external, public network is bound first, then the internal heartbeat connection. Bind remote access connections last.
- Do not rely on automatic network adapter configuration. Manually configure each adapter's driver to use the correct speed and duplex settings. Configure port switches manually as well.
- Use private IP addresses—such as 192.168.0/24—for the private network between the cluster nodes. Don't configure the private network adapters to use DHCP, DNS, or WINS; you especially don't want the private addresses being published to name resolution databases, as doing so will result in clients being unable to connect. Don't configure a default gateway; reserve that for the public adapters.
- Change the name of the network connections in Windows from Local Area Connection to Private Network and Public Network so that you can more easily tell the difference when creating the cluster.
- Once the cluster is set up, don't mess with the default Cluster Group. Don't add resources or applications to it, and don't modify the ones that are already there by default.

Above all else, I recommend purchasing cluster-compatible hardware (consult Microsoft's Cluster Hardware Compatibility List) whenever possible. Although there is no rule that says you can't configure and build your own cluster from hardware that isn't specifically listed as being cluster-compatible, doing so can be a complicated task (see the sidebar "The Myth of Specialized Cluster Hardware").

The Myth of Specialized Cluster Hardware

Microsoft has always maintained a separate Cluster Hardware Compatibility List (CHCL, now often referred to as the *Windows Catalog*), which lists hardware that has been tested and found to work well with the Windows Cluster Service. The existence of this list has given rise to the rumor that cluster hardware is somehow “special.” It’s not. In fact, you can put together a perfectly functional cluster with off-the-shelf, or *commodity*, computer parts that you pick up at a local swap meet. Of course, for reliability and performance reasons, you’ll want to purchase well-made hardware—but that doesn’t mean you have to spend a fortune.

Nearly any server on the market that supports SCSI (or can have SCSI support added) and runs Windows Server can be a cluster node. Thus, there are a wide range of options for a variety of budgets, and clusters are well within the reach of any SQL Server scale-out project.

That said, your hardware will need to provide SCSI drivers that support the Cluster Service’s special requirements, and those drivers may often require specialized installation steps, such as locating the SCSI controller card in a particular expansion slot within the server chassis. Any special requirements can often be obtained from your server vendor, but they are also one of the reasons I recommend buying preconfigured clusters when possible—to ensure that all the tricky bits are in the right place when you unpack the box.

So why bother with Microsoft’s CHCL? Support. Components and entire clusters that are on the CHCL have been tested and are known to work with clustering. Other systems and components have not been tested and might create technical difficulties. Microsoft support policies generally don’t provide you with support if you’re running clusters on non-CHCL systems or with non-CHCL components.

In this age of heightened security, it’s worth mentioning cluster security best practices:

- Don’t expose cluster members to the Internet without the use of a firewall or other protective measures.
- Do not add the Cluster Service account to the Domain Admins group or use a member of that group to start the service. It isn’t necessary.
- Do not assign a normal user account to the Cluster Service. The service’s account shouldn’t be used for interactive logons by administrators or other users.
- Applications installed in the cluster should have their own service accounts; don’t reuse the Cluster Service’s account.
- If you have more than one cluster, use a different service account for the cluster services in each cluster.
- Keep the quorum disk completely private to the cluster. Don’t share files or store application data on it.
- Don’t mess with the permissions on HKEY_LOCAL_MACHINE. Loosening security on this registry hive can give attackers an easy way to gain administrative or system permissions on cluster members.

Optimizing SQL Server Cluster Performance

Optimizing SQL Server on a cluster is pretty much the same as optimizing SQL Server on a standalone server. However, there are a few cluster-specific performance considerations:

- High-speed disk storage is a must. Rely on Fiber Channel SANs, high-performance storage, and hardware RAID and mirroring (as appropriate).
- Clustering requires shareable external storage, which can be expensive, so there is a budget-minded tendency to forget SQL Server best practices and place the data and log files onto the same partition; don't do so. Partition the external storage at the hardware level and create separate partitions for data files, log files, and so on, just like you would on a standalone server.
- Carefully plan your performance measurements and goals. If you have each node in a cluster constantly running at 90 percent utilization, performance will be slow when a node fails and the other node attempts to put out 180 percent performance. Your actual performance from an application standpoint will probably drop to something like 40 percent, instead of 50 percent, simply because the remaining node is wasting resources just trying to keep up.

Think carefully before selecting a non-Microsoft clustering solution. Although such solutions offer innovative features such as the ability to have multiple SQL Server computers accessing a single set of database files at the same time, such solutions rely on proprietary “shared clustering” techniques that Microsoft specifically chose *not* to use due to sometimes slow performance (due to database locking issues) and greater risk of damage to data. Be sure you completely understand the details of any solution you investigate.

Case Study

I worked with an Internet e-commerce company that needed to implement a scale-out solution for its SQL Server back end. The organization's business database was extremely complex and difficult to break out vertically, so they decided to go with a federation of servers and use distributed partitioned views. They settled on two servers to start, each of which contained an identical copy of their database schema and about one-quarter of the database's data. A number of distributed partitioned views were created to support their Web servers, which queried the database servers for catalog, customer, and other information.

The company also had a data warehousing application that they used for business reporting. The reporting application was important, but not considered mission-critical; they could live for several days without running reports, if necessary. They already had a server, running SQL Server 2000 (at the time), dedicated to the data warehouse.

They decided to create a 3-node cluster. Nodes A and B would each run an active instance of SQL Server 2000 and would be members of a server federation serving the Web farm that ran the company's Internet site. Node C would run a standalone, non-clustered instance of SQL Server to support the reporting application. Node C would be capable of taking over for either Node A or Node B, although doing so would limit their ability to run reports because Node C wouldn't have sufficient free resources to handle the data warehouse under those circumstances. What they built was technically an active-active-passive cluster, although the "passive" node was still performing useful, albeit non-clustered, work.

In a worst-case scenario, any one of the three servers could handle the Web farm. The servers were each built to run at about 70 percent capacity under normal conditions, so if two of them failed, the Web farm would run about 40 to 50 percent below its usual performance. But, low performance is better than a site that's completely shut down.

One clustering best practice the company decided to forego was the configuration of their cluster nodes' hardware. Nodes A and B were purchased with the intent of being cluster nodes and contained absolutely identical hardware. Node C, however, was their existing reporting server, which ran similar but not entirely identical hardware. Windows Clustering runs perfectly well under such circumstances, although you must be a bit more careful with maintenance and performance estimates because you're working with non-homogenous hardware.

The addition of the server federation boosted their overall site performance by about 20 percent; the use of clustering ensured that a single server failure wouldn't take the site completely offline. The federated database—no longer the performance bottleneck of the application—highlighted additional performance problems in the Web tier, allowing the developers to begin a project to improve performance there as well.

Interestingly, the company's original plan was to simply turn their existing single SQL Server into a 2-node, active-passive cluster. They planned to buy two servers and move the database to one of them, while using the other solely for failover purposes. I argued that this configuration was a waste of resources and suggested that an active-active cluster acting as a server federation would provide both fault tolerance and a potential performance benefit. Because their Web application had been created to use views for almost all queries, it was fairly easy to change those views to distributed partitioned views and create a distributed, partitioned database. Their desire to add fault-tolerance to their site was met, along with a significant improvement in site performance.

Database Mirroring

As mentioned earlier, database mirroring provides a high-availability capability not unlike Windows Clustering. Unlike Windows Clustering, database mirroring protects your data as well as provides failover in the event of a hardware failure. Table 6.1 will help you better understand the differences and similarities between database mirroring and Windows Clustering.

Capability	Windows Clustering	Database Mirroring
Provides redundancy for data	No	Yes
When hardware fails, all of server's databases are included in failover	Yes	No
Special hardware configuration	Yes	No
One backup server can serve for multiple production servers	Yes	Yes
Redirection to failover server automatic	Yes	Sometimes; requires specified client-side network library and connection string

Table 6.1: Comparison of Windows Clustering and database mirroring.

Database mirroring has been available for SQL Server 2000 through third-party solutions, but SQL Server 2005 is the first time Microsoft has offered this high-availability feature right out of the box. SQL Server performs mirroring by continuously sending a database's transaction log changes to a backup, or *mirror*, server. That server applies the transaction log changes, creating an identical copy of the production database. This process is essentially like transactional replication, except that no changes are made directly on the mirror and then replicated back to the production server; the "replication" is strictly one-way. Mirroring does not use SQL Server's actual replication features, even though the end result is similar; in mirroring, log changes are sent in blocks to the mirror, and the mirror's goal is to get those changes committed to disk as quickly as possible. Since the mirror isn't being used by any other processes or users—in fact, it can't be, since it's perpetually in a "recovery" state—changes can usually be committed to disk quite rapidly.

The production copy of the database is referred to as the *principal*, while the copy is, of course, called a *mirror*. Automatic failover is provided by an optional third server, called the *witness*. The witness' job is simply to watch the principal for a failure. When a failure occurs, the mirror can confirm that with the witness, and take on the role of principal—usually within a few seconds. The purpose of the witness is to help ensure that irregularities don't disrupt the network: In order for either the principal or the mirror to remain (or become) the principal, *two* servers have to agree to it. For example, if the mirror and the principal are online, they can agree that the principal is, in fact, the principal. However, if the mirror goes down, the witness can step in to confirm that the principal is still online and can remain the principal. Similarly, if the principal goes down, the witness and the mirror form a quorum and can agree that the mirror should take over as principal. The witness does *not* contain a copy of the database being mirrored.

On the client side, mirroring works best with ADO.NET or the SQL Native Client. Both of these client libraries recognize server-side mirroring and can automatically redirect if the principal changes. These libraries accept a connection string which specifies a failover partner:

```
"Data Source=A;Failover Partner=B;Initial  
Catalog=MyData;Integrated Security=True;"
```

 All editions of SQL Server 2005 can function as a witness. However, only the Enterprise, Developer, and Standard Editions can be a principal or mirror.

The principal difference—from a design perspective, at least—between Windows clustering and database mirroring is that database mirroring is configured on a per-database basis, meaning not every database on a server must be mirrored, and not every one has to use the same mirror server. Similarly, a single mirror server can contain mirrors from multiple principals. Another significant difference is in failover redirection: SQL Server 2005 requires that clients use one of two designated network libraries and a special connection string; using Windows clustering, no special client-side configuration or code is required.

 Mirroring might *sound* like a form of replication, and in some ways it works similarly to transactional replication, but it is *not* designed to mirror changes from one production database into another production database; the mirror copy can't be used for production without breaking the mirror and bringing the "hot spare" into active service. And, while replication isn't suitable for every type of database, mirroring can be used with *any* type of SQL Server database.

Summary

Although Windows Clustering doesn't offer a standalone scale-out solution, it can be an important part of an overall scale-out solution when properly used with SQL Server. Because scale-out solutions often deal with mission-critical data, Windows Clustering can offer additional fault-tolerance to your solution, helping to meet your mission-critical uptime requirement. Plus, Windows Clustering doesn't have to cost a lot more. Scale-out solutions generally involve multiple servers, so adding Windows Clustering on top of everything just provides added peace of mind. Database mirroring in SQL Server 2005 can provide similar high-availability capabilities, without the complexity often introduced by Windows Clustering.

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/>.