Realtime publishers

Tips and Tricks Guide™ To

Windows Administration

Don Jones and Dan Sullivan

Copyright Statement

Tip, Trick, Technique 23: Business Drivers Behind the Need for High Availability

There are growing needs for continuous access to business services. We expect business services, such as Web applications, database servers, email systems, and other essential business software, to be functioning and performing when we need to use them—not just when it is convenient to have them up and running. Part of this need is driven by our expectations and part is driven by the way we now design highly distributed applications.

User Expectation for Continuous Availability

Let's start by considering our expectations regarding email. Many of us check our email frequently during the day (perhaps too frequently in some cases). We have come to depend on email as a primary means of business communication. Email is so important to us that we have adopted smartphones in large numbers in order to have anytime, anywhere access to email. If you are unfortunate enough to be in an area with poor cell phone coverage and cannot access your email or can only work with it at slow speeds, you know the frustration of lack of availability.

Now consider how that same type of expectation has spread to other services such as:

- Databases
- Desktop office productivity applications
- Browsing and Internet access
- Authentication and authorization services
- File systems
- Mission-critical business applications, such as Customer Resource Management (CRM) systems and Enterprise Resource Planning (ERP) systems

It is easy to see how each of these could be essential systems in someone's work. If one or more of these systems is unavailable, productivity falls, tasks are not completed, and service level expectations are not met.

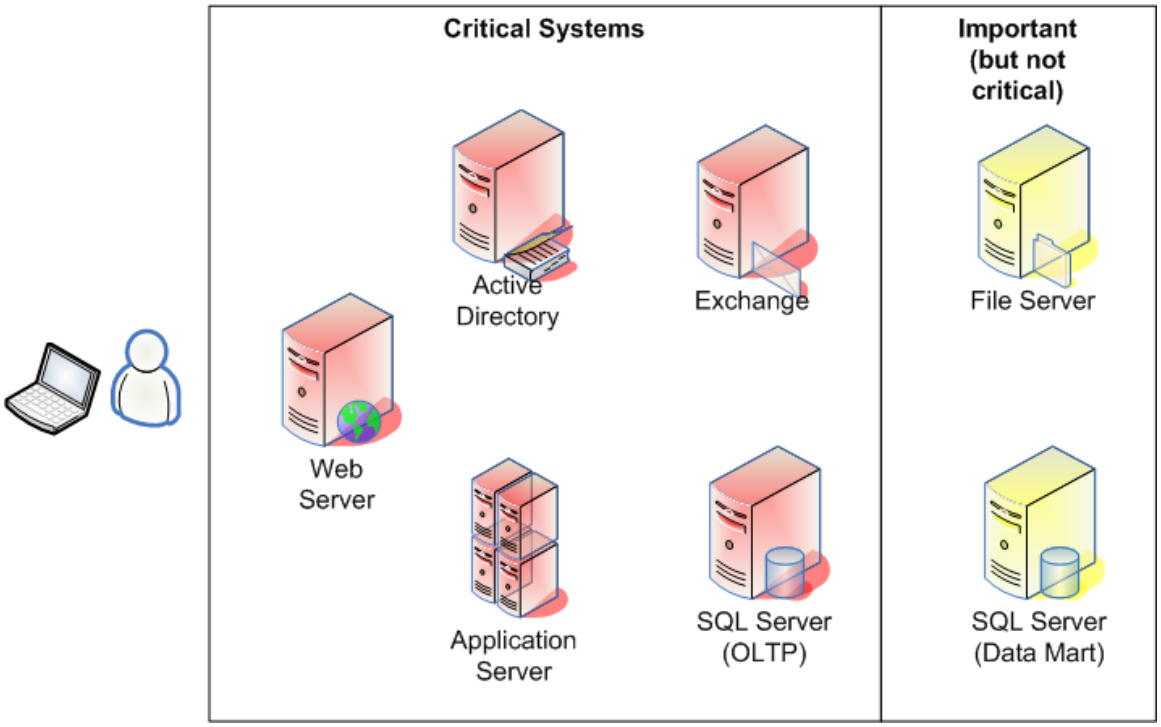

Figure 43: Maintaining availability of some systems is more important than doing so for others. Developing a high-availability strategy entails balancing costs and benefits.

Application Design Considerations and High Availability

When a customer attempts to browse your Web site and gets a timeout error instead, it is pretty obvious there is a problem. Similarly, if a Web server is up and responsive but an underlying database has failed, the problem may manifest itself with blank space where data should be listed, or worse, an error message appears about not being able to connect to the database. These server-level failures are probably the ones that are most likely to come to mind when you think of failover and high-availability solutions. There are, however, finer-grained failures you must contend with as well.

Application designs have shifted from single, tightly coupled monolithic programs to more distributed sets of software components. The Web services model of application development allows finer-grained software components to be combined in multiple ways. For example, a module for calculating tax on a customer order could be used to support multiple customer-facing Web sites.

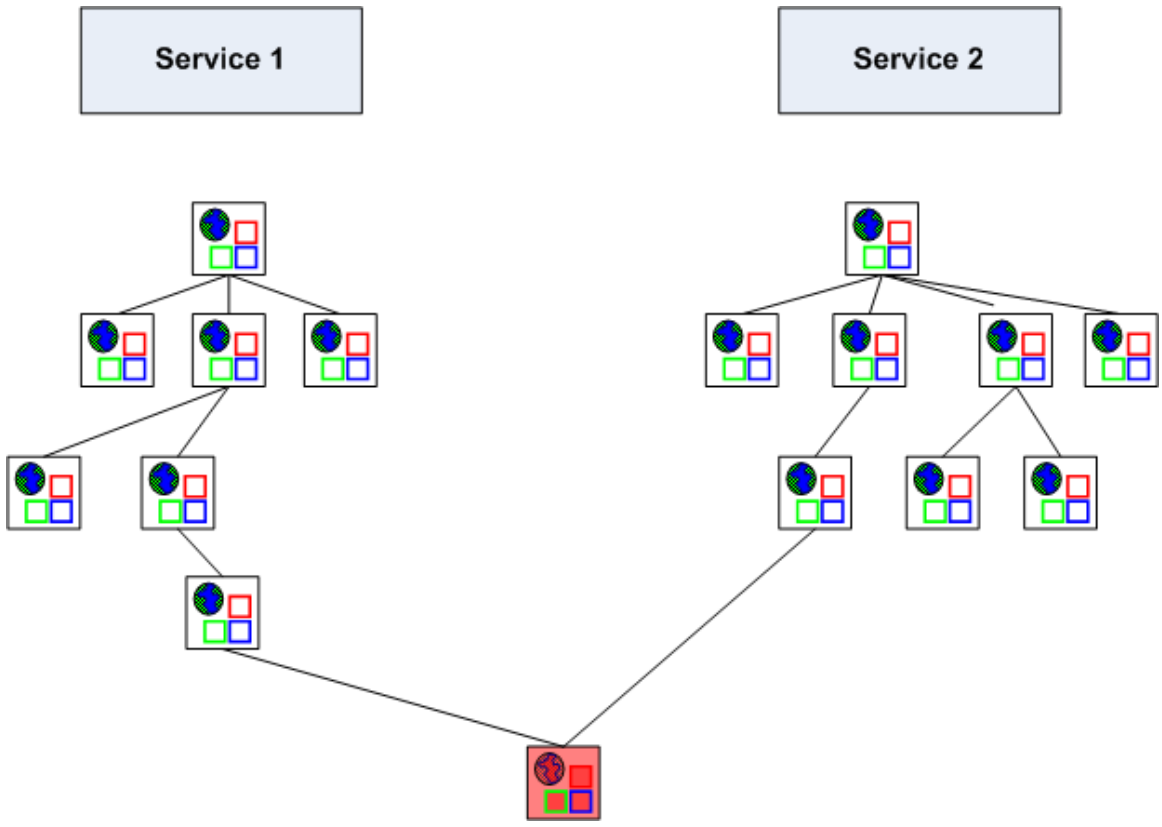

Figure 44: Applications built with Web services may have dependencies on several other services, perhaps some shared with other applications. A failure in a low-level service can have adverse effects on multiple services.

From a business perspective, it does not matter whether a system is unavailable because of a major hardware failure on a database server or there is an obscure error in a low-level Web service; any component that is critical to delivering a critical service must be available on demand.

One of the first steps in developing a high-availability strategy is mapping out the critical components that are required to provide services. Needless to say, not all services are equally important. A customer-facing support site or online catalog needs to be up continually. A shared file system used internally by employees should be up continually but if it were down for short periods of time would not cause significant adverse effects on the company. The key is to balance the cost of high availability with the benefits of continuous operations of essential business services.

Cross Reference

See Tip, Trick, Technique 18 for more information about how to identify critical systems, establish Recovery Point Objectives (RPOs), Recovery Time Objectives (RTOs), and other factors related to a comprehensive recovery management strategy.

Tip, Trick, Technique 24: Understanding the Key Elements of High Availability

High availability is the property of systems that are up and running at expected performance levels almost all the time. The definition of "almost" will vary with your requirements, but the idea is that applications, servers, and data are available to users when they expect to use those systems.

High levels of availability may allow for several minutes of downtime in a month but not much more. For example, 99.99% availability over the course of a month is equivalent to 4.32 minutes of downtime. (That is based on 60 minutes per 24 hours per 30 days or 43,200 minutes). Less than 5 minutes per month does not leave much time for patching and other maintenance operations let alone unexpected downtime. Demanding business requirements such as 99.99% availability require systems to be designed to accommodate component failures and keep working. Stringent service level agreements (SLAs) also require that services be available during maintenance, which clearly requires redundancy to provide services during planned maintenance.

The key elements of an application's architecture that you need to design for redundancy include:

- Hardware
- Operating system (OS)
- Application software

If any of these three fails, the system is potentially unavailable or unable to meet SLA requirements.

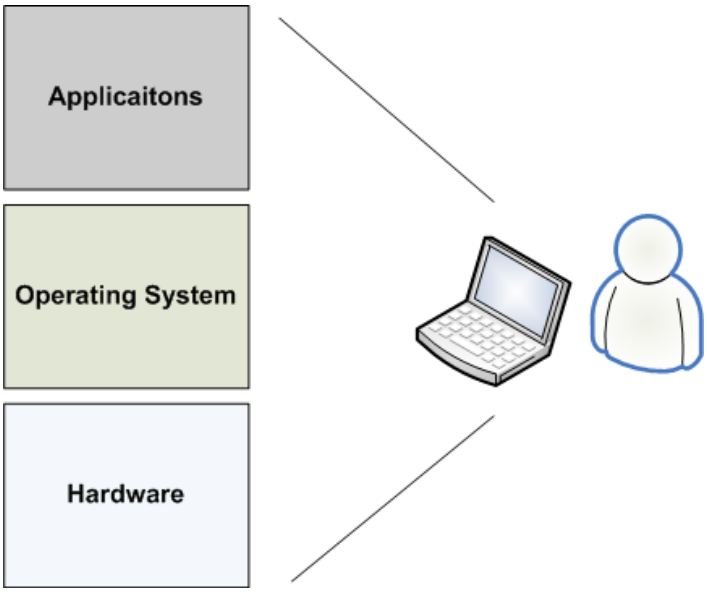

Figure 45: Service availability depends on a stack of applications, OSs, and hardware; failures in any one of these can disrupt service availability.

The Need for Hardware Redundancy

One of the simplest ways to deal with a failed component is to use another component; hardware vendors can provide us with servers with built-in redundancy of critical components. A typical list of redundant components includes:

- Power supplies
- Fans
- Network interface cards (NICs)
- Disk drives
- I/O paths to storage arrays

Fully fault-tolerant servers will also have redundant processors and memory built-in to the server. A commonly used alternative is to use multiple servers in a cluster and if a CPU fails, the workload on the primary, failed server can be switched to the standby server. The two configurations give the same level of redundancy but do so with different hardware configurations and correspondingly different methods for failure detection and workload switching.

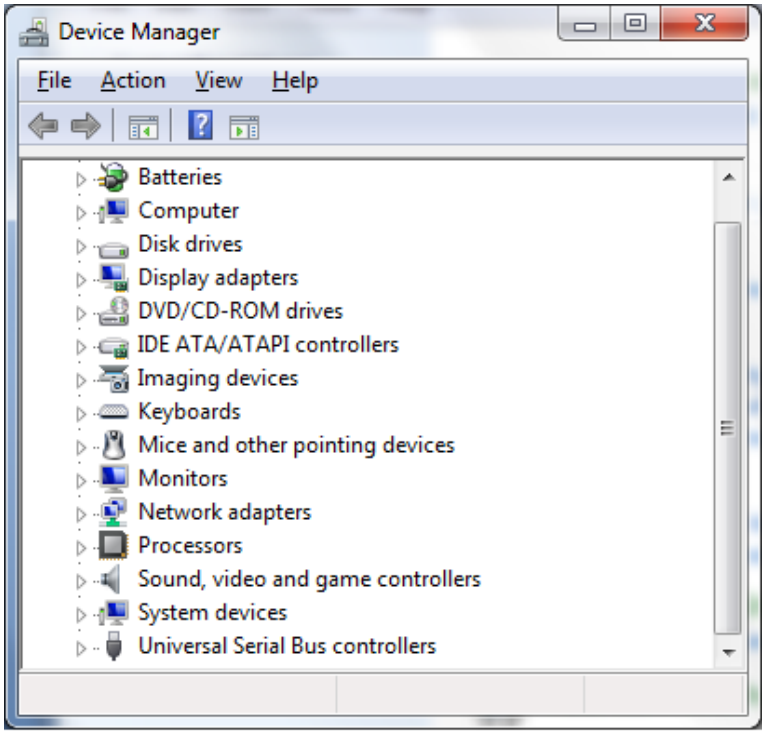

Figure 46: Any of the many hardware components of a computer can fail, but redundancy is only required for those components that could disrupt service delivery if they fail.

The Need for OS Redundancy

The Windows OSs are much more resilient today than they were in the early days of Windows NT. Systems administrators came to understand the blue-screen system dump that followed a fatal error as part of the process of maintaining the OS. Even with significant improvements in stability at the OS level, you have to plan for its failure at some point in time.

Planning for OS failure is more complex in a virtualized environment that supports multiple OSs; there is the host OS and one or more guest OSs. Recovering services requires moving guest OSs to new host OSs where they may be in the same or different combinations of virtual machines on the new host.

Special Issues with Application Software Redundancy

Redundant hardware and failover copies of the OS are two-thirds of the high-availability issue; the last piece is application software. When applications fail, you need to restart them, possibly on the same server and instance of the OS or perhaps on the same virtual machine running on a different host server. Wherever it is running, it has to recover from its failed state. How we recover from that failure depends on whether the application is stateless or stateful.

A stateless application is one that passes information from a client to a server; the server uses the information to compute a result and then returns those results. No additional information, unique to that user's session, is required. In the case of a stateful transaction, an exchange with the client can result in changes to information stored on the server, which is used to calculate results for that session. For example, an example of a stateless transaction is a Web service that accepts the name of a tax jurisdiction and a sales amount and calculates the tax. No data stored on the server about the session would influence the results. More complex services, such as one that provides an "undo" feature, may maintain state information about actions taken so that those actions could be reversed if necessary. As you shall see, the distinction between stateless and stateful applications is an important factor when choosing a high-availability option for an application.

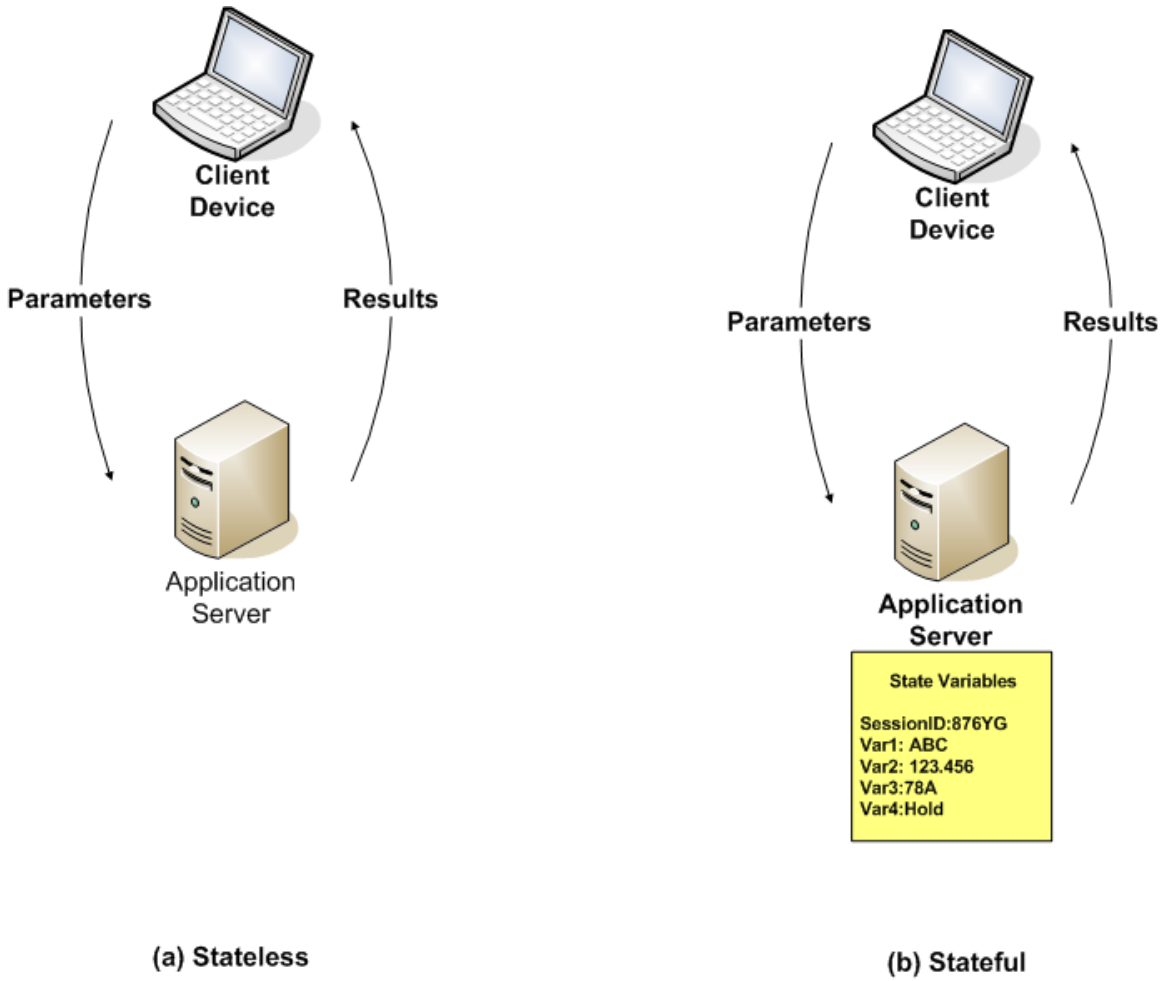

Figure 47: Stateless applications complete units of work in a single back-and-forth exchange. Stateful applications maintain information between exchanges and use that information when computing results to return to the clients.

To summarize, the key components of high availability are hardware, OSs, and application software. The need for resiliency in each of these can be addressed in multiple ways but there are constraints, ranging from the most cost-effective way to balance redundancy within and across servers to the need to support stateful sessions in Web applications. Now, let's turn our attention to Windows Server support for high availability.

Tip, Trick, Technique 25: Windows Server Options for High Availability

Windows Servers offer two methods "out of the box" for supporting high availability:

- Network Load Balancing (NLB)
- Failover clusters

Both methods use redundant servers but bring with them different advantages and disadvantages.

Using NLB to Ensure Performance Levels

NLB is a software service for distributing a workload across multiple servers. NLB runs on a Web server and allocates portions of the workload to each server in the cluster according to the NLB configuration. Tasks can be distributed in a round-robin fashion in which each new task is assigned to the next server in an ordered list that wraps around (for example, in a four server cluster, the tasks are assigned to server 1, 2, 3, 4, 1, 2, 3, and so on).

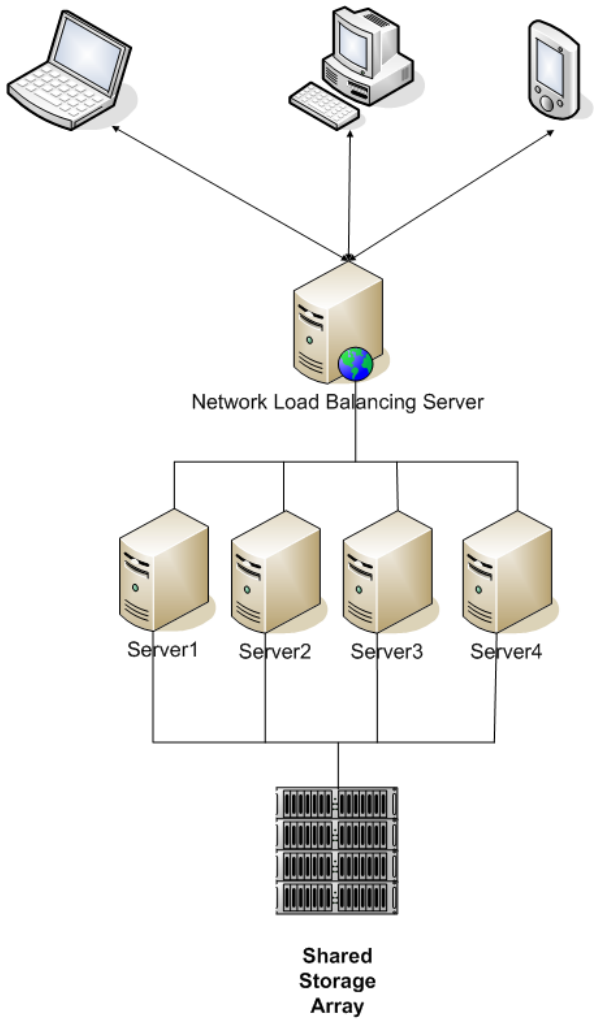

Figure 48: NLB distributes client requests over a set of clustered servers. By sharing storage, each server has access to the same persistent data.

NLB clusters such as the one depicted in Figure 48 can help maintain performance levels as demand for application services grows. Additional servers can be added to the cluster in order to meet Service Level Agreements (SLAs). They also provide failover protection; if one of the servers in an NLB cluster fails, the NLB server distributes requests to the other servers in the cluster.

NLB works well for stateless applications because each interaction between the client and server is independent. No information has to be stored on the server, therefore, one request can be serviced by one server and the next request can be serviced by another.

An NLB cluster can be used for applications that maintain state information if the application writes that data to a shared, persistent data store, preferably a database. In this case, however, you introduce a potential single point of failure unless the database itself is set up in a high-availability configuration. SQL Server databases can be configured in failover clusters, which would address this issue. Failover clusters a different type of cluster than NLB clusters. The former provides redundancy for running an application; the latter is used to distribute TCP/IP traffic among multiple servers.

Application Redundancy in Failover Clusters

Failover clusters eliminate single points of failure with regard to servers by having standby servers in place and ready to assume the function of the primary servers in case the primary servers become unavailable. Failover clusters can be configured in one of two ways: Active/Active and Active/Passive.

The Active/Active model is similar to NLB clusters in that all the nodes typically share in the workload at all times. In the case of Active/Passive clusters, one of the servers is the primary server, which handles all the workload while the standby server remains passive. When the failover software detects the primary is no longer available, the passive server becomes active and starts serving requests.

Failover clusters are configured with multiple servers. In the simplest case of two servers, the passive server would only assume responsibility for providing application services if the active server is no longer available. (A monitoring mechanism known as a heartbeat is used to detect when a server fails. More on the complexities of detecting failures in a minute). When more than two servers are in a failover cluster, you need a mechanism to ensure that the two passive servers do not both attempt to assume the role of the active server. To prevent multiple active servers, a quorum model is used.

The quorum model is a voting system in which each server has a vote and the storage system may have a vote as well. A majority of votes is needed to elevate a passive server to an active server.

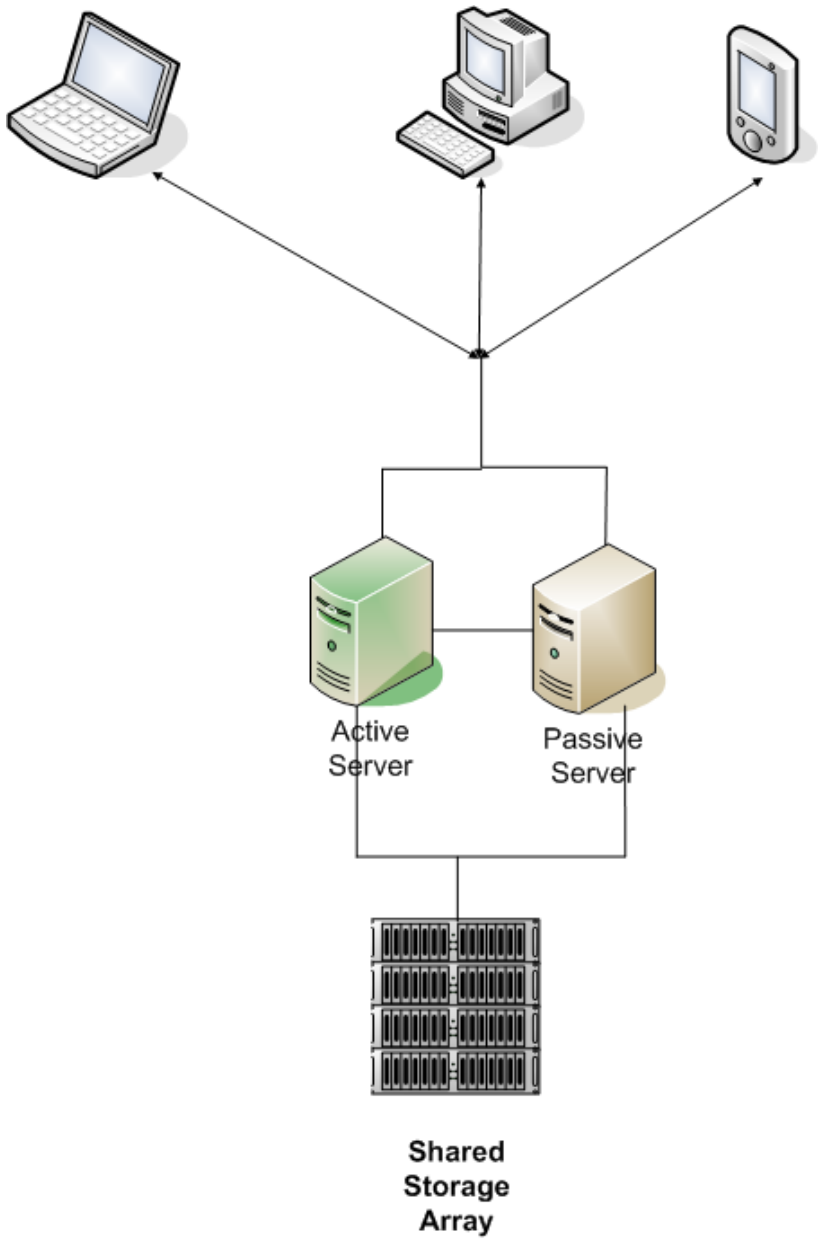

Figure 49: Failover clusters use multiple servers with the ability to detect failure in other servers. If the active server were to fail, the passive server would take on the workload of the failed server.

Windows Server 2008 introduced a number of features in failover clusters that ease the job of deploying and managing clusters:

- Failover Cluster Management snap-in for the Microsoft Management Console (MMC)
- High-Availability Wizard to assist with setup
- Cluster validation tool to check whether a hardware configuration will support failover clusters
- Improved support for multi-site clusters, which is especially helpful for disaster recovery configurations

Both NLB and failover clusters provide application high availability. NLB is well suited for stateless applications while failover clusters work well for providing resiliency in other types of services, such as file and print services or applications supporting clustered environments, such as Microsoft Exchange and Microsoft SQL Server.

Tip, Trick, Technique 26: Ensuring High Availability for SQL Server Databases

Up to this point, the focus has been on general approaches to high availability. In this Tip, Trick, Technique, we focus on ensuring high availability with SQL Server databases. Databases have characteristics—such as I/O and compute-intensive operations—not found in other applications. As databases are designed to store persistent data, high availability has to address the need for saving data as well as restarting computations after the failure of a primary server.

As of SQL Server 2008, there are several options for providing high-availability databases:

- Failover clusters
- Database mirroring
- Log shipping
- Replication

Not surprisingly, there are tradeoffs between functionality and ease of management. Depending on your requirements, one or more of these solutions may be more appropriate than the others.

Failover Clusters

A failover cluster is a set of two or more servers that support a single instance of a SQL Server database. An *instance* is database parlance for a single managed database that users can connect to for persistent data storage and management services. The database instance is installed on a cluster of servers, known as a *resource group*. A resource group is a set of servers that logically function as a single computing resource. All servers in the group share storage.

One of the advantages of failover clusters is that servers in the resource group can detect whether the server running the database has failed. If so, one of the other members of the group assumes responsibility for providing database services. Another advantage is that the resource group appears as a single server, so applications connect to the group rather than a specific server. With this method, there is no need to change connection information to redirect applications to another server.

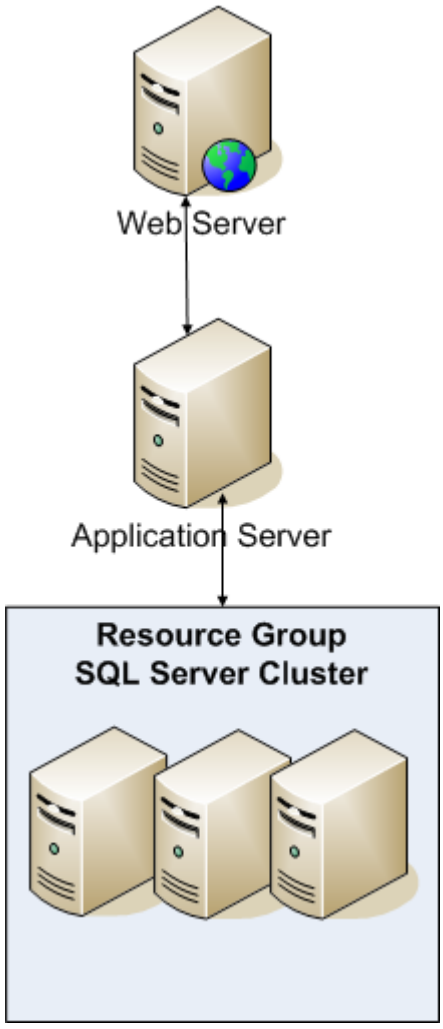

Figure 50: A SQL Server failover cluster is a set of servers that function as a single database server; when a server fails, another server in the resource group assumes its function.

Database Mirroring

Another way to improve availability with SQL Server 2005 and 2008 is database mirroring. The basic idea is that every change applied to a primary database is also sent to a backup database. For example, if a user executes a SQL statement such as:

```
UPDATE sales.customer
SET active = 'Y'
WHERE cust_id = 78973;
```

The customer record will be updated on the primary server, also known as the *principal server*. The same transaction is sent to the secondary, or mirrored, server for execution as well.

One consideration with database mirroring is when to commit a transaction on the primary server. In synchronous mode, also known as high safety mode, the transaction is sent to the secondary server and the primary server waits until that transaction completes before completing the transaction on the primary server. This setup guarantees the update is written to the secondary server before the primary considers the transaction complete.

In asynchronous mode, also known as high performance mode, the primary sends the transaction to the secondary server but does not wait for verification that the transaction completes. It is conceivable that a transaction successfully performed on the primary server fails on the secondary. (There could be insufficient space on the secondary, for example). The tradeoff here is that the primary does not have to wait for both transactions to complete before it can move on to the next operation.

Log Shipping

Log shipping is another technique for duplicating transactions on a secondary server. Unlike database mirroring where individual transactions are sent to the secondary server, with log shipping, a backup of the transaction log on the primary is copied and sent to the secondary server where it is applied.

There is a potential to lose transactions with this method. For example, if a log is shipped from the primary to the secondary server and then other transactions are applied to the primary and the primary fails, those later transactions will not be recovered. In spite of this limitation, log shipping may be suitable for low transaction databases, such as reporting systems that are replicated to improve performance.

Replication

Replication is a way of duplicating data using a publish and subscribe model. SQL Server 2008 provides three types of replication:

- Transactional replication
- Snapshot replication
- Merge replication

Transactional replication is useful for high-availability applications where a steady stream of transactions need to be copied from one server to another. Snapshot replication is useful for copying data *en masse* from one server to another; for example, to a make a secondary copy that is then maintained with transactional replication. Merge replication is more useful for distributed applications that may result in conflicts when data is updated, such as mobile applications.

Clearly there are many options for ensuring high availability with SQL Server. Failover clusters provide computing redundancy with shared storage. If you are more concerned with servers failing than a highly reliable disk array, failover clusters may be an appropriate choice. If you are looking for disaster recovery features as well as high availability, you will be running two storage systems as well. Data mirroring, log shipping, and replication with a publish and subscribe method can be used in that case.

Tip, Trick, Technique 27: Ensuring High Availability for Microsoft Exchange

When it comes to ensuring high availability of email services, your options depend on the version of Microsoft Exchange you are running. Microsoft Exchange 2007 had a few options, including a failover cluster option; with the advent of Microsoft Exchange 2010, high availability was built into Microsoft Exchange itself. The later version of the email server made high availability less cumbersome and easier to manage. The two versions' approaches to replication are so different, we will consider them separately.

High Availability in Microsoft Exchange 2007

Microsoft Exchange 2007 offered email administrators a few options for implementing high-availability solutions:

- Local continuous replication
- Single copy clusters
- Clustered mailbox servers

Local continuous replication uses asynchronous replication to create copies of data that are then kept synchronized using transaction log shipping. The copies are kept on a second set of disks on the same server. This option does not provide server failover protection but does keep a second copy of data, which can help if the primary copy is corrupted or otherwise unusable.

Single copy clusters are built on failover clusters. As with other failover clusters, the servers within the cluster share a single storage system. Configuring these clusters and maintaining them can be challenging and time consuming.

Clustered mailbox servers use clustering technology for server failover but do not use shared storage. Instead, a copy of the mail data is maintained with asynchronous log shipping. This option has both multiple servers and multiple copies of data, so it can provide a recovery path for both server and data failure. If the servers are located in different data centers, this option provides disaster recovery protection as well.

High Availability in Microsoft Exchange 2010

Microsoft Exchange 2010 makes a break with the failover cluster model used in Microsoft Exchange 2007 and instead builds high-availability features into the application itself. Some of the new features of Exchange 2010 include:

- Actively pushing log data from primary to secondary servers
- Continuous replication is done at the database level
- Improved protection for messages in transit

The result is improved protection and easier management.

This is a trend of pushing availability features into the application level that is likely to continue as high availability moves from low-level network/server function to higher-level business process and application level.

Download Additional Books from Realtime Nexus!

Realtime Nexus—The Digital Library provides world-class expert resources that IT professionals depend on to learn about the newest technologies. If you found this book to be informative, we encourage you to download more of our industry-leading technology books and video guides at Realtime Nexus. Please visit http://nexus.realtimepublishers.com.