Server Clusters : Architecture Overview

For Windows Server 2003

Microsoft Corporation

Published: March 2003

Abstract

Server clusters are one of two Microsoft® Windows® clustering technologies available for the Microsoft Windows Server family of products. Windows Server 2003 clusters provide failover support for back-end applications and services that require high availability and data integrity. These back-end applications include enterprise applications such as database, file server, enterprise resource planning (ERP), and messaging systems. This white paper focuses on the architecture and features of the cluster service and describes its terminology, concepts, design goals, key components, and planned future directions.

Content

Introduction

First designed for the Microsoft Windows NT® Server 4.0 operating system, server clusters are substantially enhanced in the Microsoft Windows Server 2003, Enterprise Edition, and Windows Server 2003, Datacenter Edition, operating systems. With server clusters you can connect multiple servers together in order to provide high availability and easy manageability of data and programs running within the cluster. Server clusters provide the following three principal advantages in clustering technology:

- **Improved availability** by enabling services and applications in the server cluster to continue providing service during hardware or software component failure or during planned maintenance.
- **Increased scalability** by supporting servers that can be expanded with the addition of multiple processors (up to a maximum of eight processors in Windows Server 2003, Enterprise Edition, and 32 processors in Windows Server 2003, Datacenter Edition), and additional memory (up to a maximum of 8 gigabytes [GB] of random access memory [RAM] in Enterprise Edition and 64 GB in Windows Server 2003 Datacenter Edition).
- **Improved manageability** by enabling administrators to manage devices and resources within the entire cluster as if they were managing a single computer.

The cluster service is one of two complementary Windows clustering technologies provided as extensions to the base Windows Server 2003 and Windows 2000 operating systems. The other clustering technology, Network Load Balancing (NLB), complements server clusters by supporting highly available and scalable clusters for front-end applications and services such as Internet or intranet sites, Web–based applications, media streaming, and Microsoft Terminal Services.

This white paper focuses solely on the architecture and features of server clusters and describes the terminology, concepts, design goals, key components, and planned future directions. At the end of the paper, the section “For More Information” provides a list of references you can use to learn more about server clusters and the NLB technologies.

Development Background

Computer clusters have been built and used for well over a decade. One of the early architects of clustering technology, G. Pfister, defined a cluster as "a parallel or distributed system that consists of a collection of interconnected whole computers that is utilized as a single, unified computing resource."

The collection of several server computers into a single unified cluster makes it possible to share a computing load without users or administrators needing to know that more than one server is involved. For example, if any resource in the server cluster fails the cluster as a whole can continue to offer service to users by using a resource on one of the other servers in the cluster, regardless of whether the failed component is a hardware or software resource.

In other words, when a resource fails, users connected to the server cluster may experience temporarily degraded performance, but do not completely lose access to the service. As more processing power is needed, administrators can add new resources in a rolling upgrade process. The cluster as a whole remains online and available to users during the process, while the post-upgrade performance of the cluster improves.

User and business requirements for clustering technology shaped the design and development of the cluster service for the Windows Server 2003, Enterprise Edition, and Windows Server, 2003 Datacenter Edition, operating systems. The principal design goal was development of an operating system service that addressed the cluster needs of a broad segment of businesses and organizations, rather than small, specific, market segments.

Microsoft marketing studies showed a large and growing demand for high availability systems in small- and medium-sized businesses as databases and electronic mail became essential to their daily operations. Ease of installation and management were identified as key requirements for organizations of this size. At the same time, Microsoft's research showed an increasing demand for Windows–based servers in large enterprises with key requirements for high performance and high availability.

The market studies led to the development of the cluster service as an integrated extension to the base Windows NT, Windows 2000, and Windows Server 2003 operating systems. As designed, the service enables joining multiple server and data storage components into a single, easily managed unit, the *Server cluster*. Server clusters can be used by small and large enterprises to provide highly available and easy-to-manage systems running Windows Server 2003 and Windows 2000–based applications. Server clusters also provide the application interfaces and tools needed to develop new, "cluster-aware" applications that can take advantage of the high availability features of server clusters.

Cluster Terminology

Server clusters is the Windows Server 2003 name for the Microsoft technology first made available as Microsoft Cluster Server (MSCS) in Windows NT Server 4.0, Enterprise Edition. When referring to servers that comprise a cluster, individual computers are referred to as *nodes*. The *cluster service* refers to the collection of components on each node that perform cluster-specific activity and r*esource* refers to the hardware and software components within the cluster that are managed by the cluster service. The instrumentation mechanism provided by server clusters for managing resources is the resource dynamically linked libraries (DLLs). Resource DLLs define resource abstractions, communication interfaces, and management operations.

A resource is said to be *online* when it is available and providing its service to the cluster. Resources are physical or logical entities if they:

- Can be brought online (in service) and taken offline (out of service).
- Can be managed in a server cluster.
- Can be owned by only one node at a time.

Cluster resources include physical hardware devices such as disk drives and network cards, and logical items such as Internet Protocol (IP) addresses, applications, and application databases. Each node in the cluster will have its own local resources. However, the cluster also has common resources, such as a common data storage array and private cluster network. These common resources are accessible by each node in the cluster. One special common resource is the *quorum resource*, a physical disk in the common cluster disk array that plays a critical role in cluster operations. It must be present for node operations—such as forming or joining a cluster—to occur.

A *resource group* is a collection of resources managed by the cluster service as a single, logical unit. Application resources and cluster entities can be easily managed by grouping logically related resources into a resource group. When a cluster service operation is performed on a resource group, the operation affects all individual resources contained within the group. Typically, a resource group is created to contain all the elements needed by a specific application server and client for successful use of the application.

Server Clusters

Server clusters are based on a *shared-nothing* model of cluster architecture. This model refers to how servers in a cluster manage and use local and common cluster devices and resources. In the shared-nothing cluster, each server owns and manages its local devices. Devices common to the cluster, such as a common disk array and connection media, are selectively owned and managed by a single server at any given time.

The shared-nothing model makes it easier to manage disk devices and standard applications. This model does not require any special cabling or applications and enables server clusters to support standard Windows Server 2003 and Windows 2000–based applications and disk resources.

Server clusters use the standard Windows Server 2003 and Windows 2000 Server drivers for local storage devices and media connections. Server clusters support several connection media for the external common devices that need to be accessible by all servers in the cluster. External storage devices that are common to the cluster require small computer system interface (SCSI) devices and support standard PCI–based SCSI connections as well as SCSI over Fibre Channel and SCSI bus with multiple initiators. Fibre connections are SCSI devices, simply hosted on a Fibre Channel bus instead of a SCSI bus. Conceptually, Fibre Channel technology encapsulates SCSI commands within the Fibre Channel and makes it possible to use the SCSI commands which server clusters are designed to support. These SCSI commands (Reserve/Release and Bus Reset) will function the same over standard or non-fibre SCSI interconnect media.

The following figure illustrates components of a two-node server cluster that may be composed of servers running either Windows Server 2003, Enterprise Edition, or Windows 2000, Advanced Server, with shared storage device connections using SCSI or SCSI over Fibre Channel.

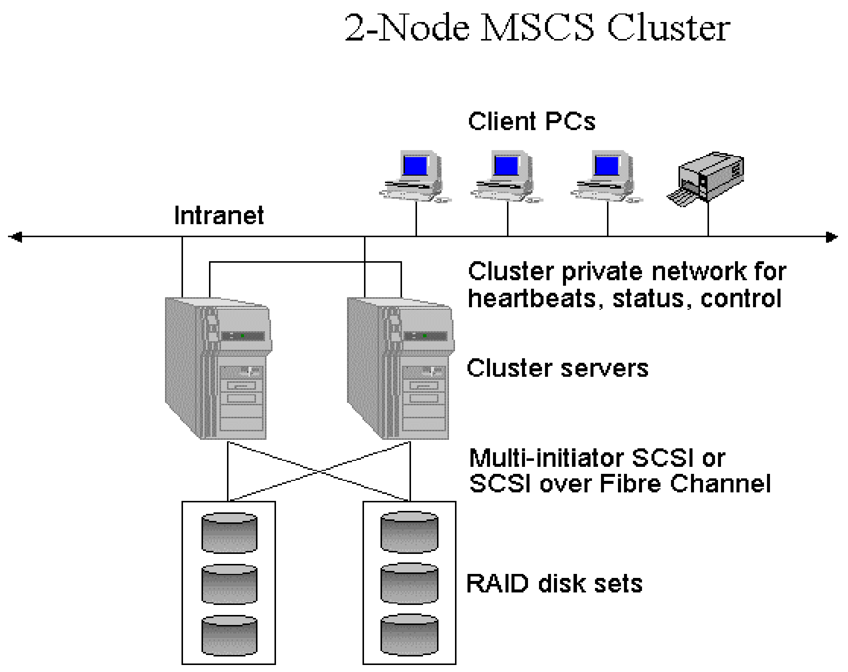
2-Node MSCS Cluster
Client PCs
Intranet
Cluster private network for heartbeats, status, control
Cluster servers
Multi-initiator SCSI or SCSI over Fibre Channel
RAID disk sets

Figure 1 - Two-node Server cluster running Windows Server 2003, Enterprise Edition

Windows Server 2003 Datacenter Edition supports four- or eight-node clusters and does require device connections using Fibre Channel as shown in the following illustration of the components of a four-node cluster.

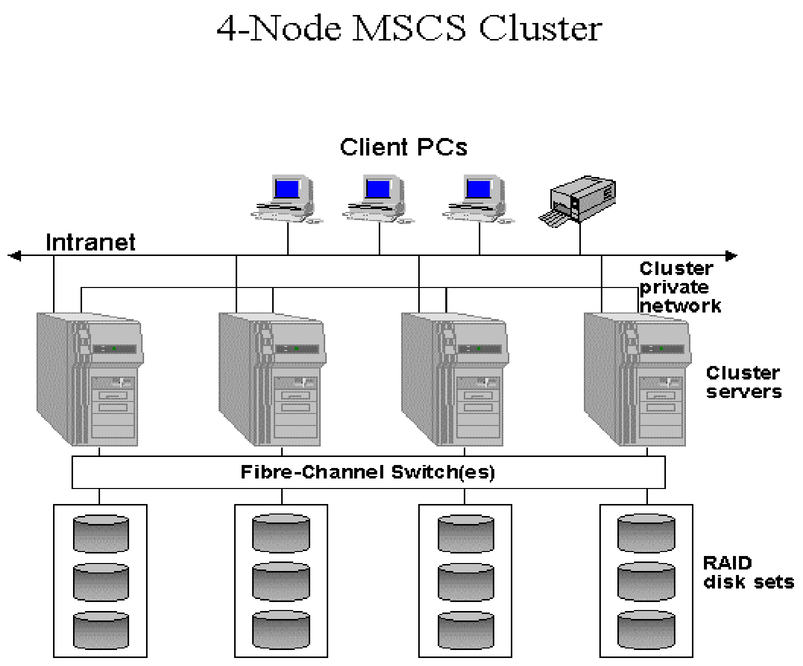

Figure 2 - Four-node Server cluster running Windows Server 2003 Datacenter Edition

Virtual Servers

One of the benefits of clusters is that applications and services running on a server cluster can be exposed to users and workstations as virtual servers. To users and clients, connecting to an application or service running as a clustered virtual server appears to be the same process as connecting to a single, physical server. In fact, the connection to a virtual server can be hosted by any node in the cluster. The user or client application will not know which node is actually hosting the virtual server.

Note: Services or applications that are not accessed by users or client applications can run on a cluster node without being managed as a virtual server.

Multiple virtual servers representing multiple applications can be hosted in a cluster. This is illustrated in Figure 3.

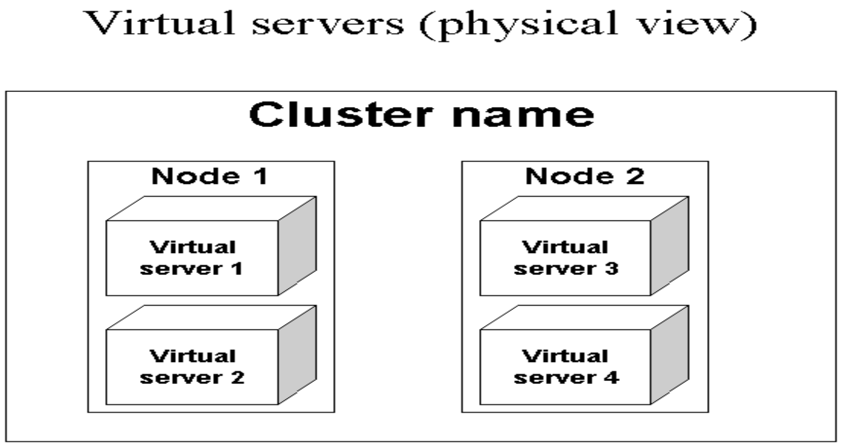

Figure 3 - Physical view of virtual servers under server clusters

The figure above illustrates a two-node cluster with four virtual servers; two virtual servers exist on each node. Server clusters manage the virtual server as a resource group, with each virtual server resource group containing two resources: an IP address and a network name that is mapped to the IP address.

Application client connections to a virtual server are made by a client session that knows only the IP address that the cluster service publishes as the address of the virtual server. The client view is simply a view of individual network names and IP addresses. Using the example of a 2-node cluster supporting four virtual servers, the client view of the cluster nodes and four virtual servers is illustrated in Figure 4.

As shown in Figure 4, the client only sees the IP addresses and names and does not see information about the physical location of any of the virtual servers. This allows server clusters to provide highly available support for the applications running as virtual servers.

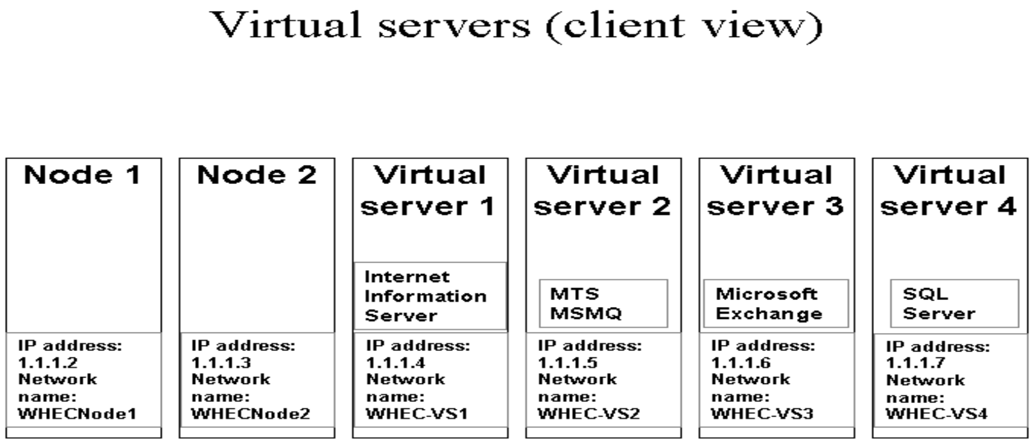
Virtual servers (client view)
Node 1
IP address: 1.1.1.2 Network name: WHECNode1
Node 2
IP address: 1.1.1.3 Network name: WHECNode2
Virtual server 1
Internet Information Server
IP address: 1.1.1.4 Network name: WHEC-VS1
Virtual server 2
MTS MSMQ
IP address: 1.1.1.5 Network name: WHEC-VS2
Virtual server 3
Microsoft Exchange
IP address: 1.1.1.6 Network name: WHEC-VS3
Virtual server 4
SQL Server
IP address: 1.1.1.7 Network name: WHEC-VS4

Figure 4 - Client view of server cluster virtual servers

In the event of an application or server failure, the cluster service moves the entire virtual server resource group to another node in the cluster. When such a failure occurs, the client will detect a failure in its session with the application and attempt to reconnect in exactly the same manner as the original connection. It will be able to do this successfully, because the cluster service simply maps the published IP address of the virtual server to a surviving node in the cluster during recovery operations. The client session can reestablish the connection to the application without needing to know that the application is now physically hosted on a different node in the cluster.

Note that while this provides high availability of the application or service, session state information related to the failed client session is lost unless the application is designed or configured to store client session data on a disk for retrieval during application recovery. Server clusters enable high availability, but do not provide application fault tolerance, unless the application itself supports fault-tolerant transaction behavior. Microsoft Dynamic Host Configuration Protocol (DHCP) service is a service that provides an example of an application that stores client data and can recover from failed client sessions. DHCP client IP address reservations are saved in the DHCP database. If the DHCP server resource fails, the DHCP database can be moved to an available node in the cluster, and restarted with restored client data from the DHCP database.

Resource Groups

Resource groups are logical collections of cluster resources. Typically a resource group is made up of logically related resources such as applications and their associated peripherals and data. However, resource groups can contain cluster entities that are related only by administrative needs, such as an administrative collection of virtual server names and IP addresses. A resource group can be owned by only one node at a time and individual resources within a group must exist on the node that currently owns the group. At any given instance, different servers in the cluster cannot own different resources in the same resource group.

Each resource group has an associated cluster-wide policy that specifies which server the group prefers to run on, and which server the group should move to in case of a failure. Each group also has a network service name and address to enable network clients to bind to the services provided by the resource group. In the event of a failure, resource groups can be failed over or moved as atomic units from the failed node to another available node in the cluster.

Each resource in a group may depend on other resources in the cluster. Dependencies are relationships between resources that indicate which resources need to be started and available before another resource can be started. For example, a database application may depend on the availability of a disk, IP address, and network name to be able to start and provide services to other applications and clients.

Resource dependencies are identified using cluster resource group properties and enable the cluster service to control the order in which resources are brought on- and off- line. The scope of any identified dependency is limited to resources within the same resource group. Cluster-managed dependencies cannot extend beyond the resource group, because resource groups can be brought online and offline and moved independently.

Server Cluster Architecture

Server clusters are designed as a separate, isolated set of components that work together with the operating system. This design avoids introducing complex processing system schedule dependencies between the server clusters and the operating system. However, some changes in the base operating system are required to enable cluster features. These changes include:

- Support for dynamic creation and deletion of network names and addresses.
- Modification of the file system to enable closing open files during disk drive dismounts.
- Modifying the input/output (I/O) subsystem to enable sharing disks and volume sets among multiple nodes.

Apart from the above changes and other minor modifications, cluster capabilities are built on top of the existing foundation of the Windows Server 2003 and Windows 2000 operating systems.

The core of server clusters is the cluster service itself, which is composed of several functional units. These include the Node Manager, Failover Manager, Database Manager, Global Update Manager, Checkpoint Manager, Log Manager, Event Log Replication Manager, and the Backup/Restore Manager. A high level architectural diagram of the relationship between these components is shown in Figure 6.

Cluster Service Components

The cluster service runs on the Windows Server 2003 or Windows 2000 operating system using network drivers, device drivers, and resource instrumentation processes designed specifically for server clusters and its component processes. These closely related, cooperating components of the cluster service are:

- **Checkpoint Manager**—saves application registry keys in a cluster directory stored on the quorum resource.
- **Database Manager—**maintains cluster configuration information.
- **Event Log Replication Manager**—replicates event log entries from one node to all other nodes in the cluster.
- **Failover Manager**—performs resource management and initiates appropriate actions, such as startup, restart, and failover.
- **Global Update Manager—**provides a global update service used by cluster components.
- **Log Manager**—writes changes to recovery logs stored on the quorum resource.
- **Membership Manager—**manages cluster membership and monitors the health of other nodes in the cluster.
- **Node Manager—**assigns resource group ownership to nodes based on group preference lists and node availability.
- **Resource Monitors—**monitors the health of each cluster resource using callbacks to

resource DLLs. Resource Monitors run in a separate process, and communicate with the Cluster Server through remote procedure calls (RPCs) to protect cluster server from individual failures in cluster resources.

- **Backup/Restore Manager—**backs up, or restores, quorum log file and all checkpoint files, with help from the Failover Manager and the Database Manager.

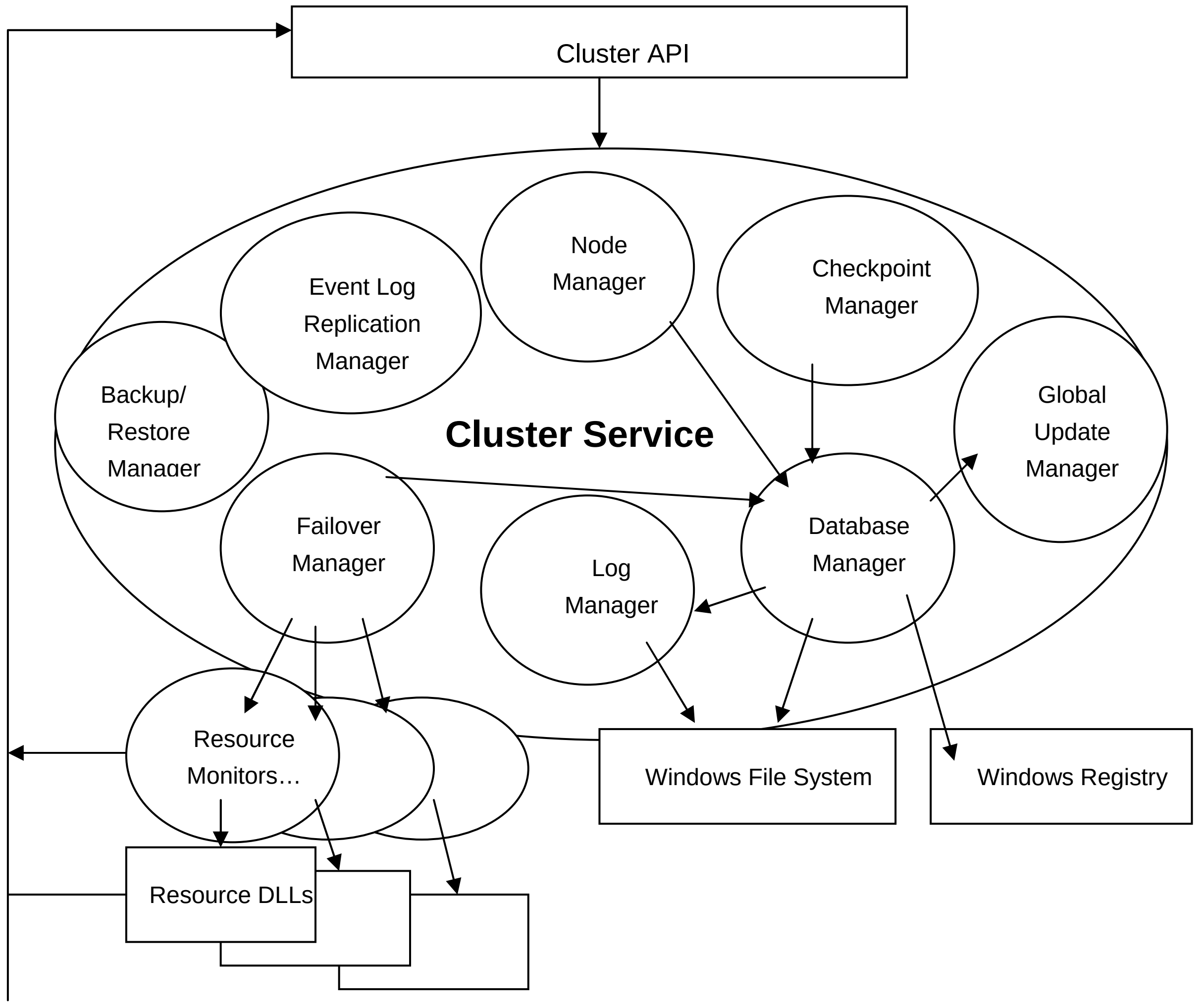

Figure 5 - Diagram of cluster service components

Node Manager

The Node Manager runs on each node and maintains a local list of nodes that belong to the cluster. Periodically, the Node Manager sends messages—called *heartbeats*—to its counterparts running on other nodes in the cluster to detect node failures. It is essential that all nodes in the cluster always have exactly the same view of cluster membership.

In the event that one node detects a communication failure with another cluster node, it multicasts a message to the entire cluster, causing all members to verify their view of the current cluster membership. This is called a *regroup event*. The cluster service prevents write operations to any disk devices common to all nodes in the cluster until the membership has stabilized. If the Node Manager on an individual node does not respond, the node is removed from the cluster and its active resource groups are moved to another active node. To select the node to which a resource group should be moved, Node Manager identifies the node on which a resource group prefers to run and the possible owners (nodes) that may own individual resources. On a two-node cluster, the Node Manager simply moves resource groups from a failed node to the surviving node. On a cluster with three or more nodes, Node Manager selectively distributes resource groups among the surviving nodes.

Node Manager also acts as a gatekeeper, allowing “joiner” nodes into the cluster, as well as processing requests to add or evict a node.

Note: In the event that the cluster service and its component processes should fail, resources attached to the node experiencing the failure are stopped under the assumption that they will be restarted on an active node in the cluster.

Database Manager

The Database Manager provides the functions needed to maintain the cluster configuration database, which contains information about all of the physical and logical entities in a cluster. These entities include the cluster itself, cluster node membership, resource groups, resource types, and descriptions of specific resources, such as disks and IP addresses.

Persistent and volatile information stored in the configuration database is used to track the current and desired state of the cluster. Each Database Manager running on each node in the cluster cooperates to maintain consistent configuration information across the cluster. One-phase commits are used to ensure the consistency of the copies of the configuration database on all nodes. The Database Manager also provides an interface for use by the other cluster service components, such as the Failover Manager and the Node Manager. This interface is similar to the registry interface exposed by the Win32 application programming interface (API) set. The key difference is that changes made to cluster entities are recorded by the Database Manager in the both the registry and in the quorum resource (changes are written to the quorum resource by the Log Manager). Registry changes are then replicated to other nodes by the Global Update Manager.

The Database Manager supports transactional updates of the cluster hive and exposes the interfaces only to internal cluster service components. This transactional support is typically used by the Failover Manager and the Node Manager in order to get replicated transactions.

Database Manager functions, with the exceptions of those for transactional support, are exposed to clients by the cluster API.   The primary clients for these Database Manager APIs are resource DLLs that use the Database Manager to save private properties to the cluster database. Other clients typically use the Database Manager to query the cluster database.

Note: Application registry key data and changes are recorded by the Checkpoint Manager in quorum log files on the quorum resource.

Checkpoint Manager

To ensure that the cluster service can recover from a resource failure, the Checkpoint Manager checks registry keys when a resource is brought online and writes checkpoint data to the quorum resource when the resource goes offline. Cluster-aware applications use the cluster configuration database to store recovery information. Applications that are not cluster-aware store information in the local server registry.

The Checkpoint Manager also supports resources having application-specific registry trees instantiated at the cluster node where the resource comes online (a resource can have one or more registry trees associated with it). The Checkpoint Manager watches for changes made to these registry trees if the resource is online. If it detects that changes have been made, it creates a dump of the registry tree on the owner node of the resource and then moves the file to the owner node of the quorum resource. The Checkpoint Manager performs some amount of “batching” so that frequent changes to registry trees do not place too heavy a load on the cluster service.

Log Manager

The Log Manager, along with the Checkpoint Manager, ensures that the recovery log on the quorum resource contains the most recent configuration data and change checkpoints. If one or more cluster nodes are down, configuration changes can still be made to the surviving nodes. While these nodes are down, the Database Manager uses the Log Manager to log configuration changes to the quorum resource.

As the failed nodes return to service, they read the location of the quorum resource from their local cluster hives. Since the hive data could be stale, mechanisms are built in to detect invalid quorum resources that are read from a stale cluster configuration database. The Database Manager will then request the Log Manager to update the local copy of the cluster hive using the checkpoint file in the quorum resource, and then replay the log file in the quorum disk starting from the checkpoint log sequence number. The result is a completely updated cluster hive.

Cluster hive snapshots are taken whenever the quorum log is reset and once every four hours.

Failover Manager

The Failover Manager is responsible for stopping and starting resources, managing resource dependencies, and for initiating failover of resource groups. To perform these actions, it receives resource and system state information from Resource Monitors and the cluster node.

The Failover Manager is also responsible for deciding which nodes in the cluster should own which resource group. When resource group arbitration finishes, nodes that own an individual resource group turn control of the resources within the resource group over to Node Manager. When failures of resources within a resource group cannot be handled by the node that owns the group, Failover Managers on each node in the cluster work together to re-arbitrate ownership of the resource group.

If a resource fails, Failover Manager might restart the resource, or take the resource offline along with its dependent resources. If it takes the resource offline, it will indicate that the ownership of the resource should be moved to another node and be restarted under ownership of the new node. This is referred to as *failover*.

Failover

Failover can occur automatically because of an unplanned hardware or application failure, or can be triggered manually by the person who administers the cluster. The algorithm for both situations is identical, except that resources are shut down in an orderly fashion for a manually initiated failover, while their shut down may be sudden and disruptive in the failure case.

When an entire node in a cluster fails, its resource groups are moved to one or more available servers in the cluster. Automatic failover is similar to planned administrative reassignment of resource ownership. It is, however, more complicated, because the orderly steps of a normal shutdown may have been interrupted or may not have happened at all. As a result, extra steps are required in order to evaluate the state of the cluster at the time of failure.

Automatic failover requires determining what groups were running on the failed node and which nodes should take ownership of the various resource groups. All nodes in the cluster that are capable of hosting the resource groups negotiate among themselves for ownership. This negotiation is based on node capabilities, current load, application feedback, or the *node preference list*. The node preference list is part of the resource group properties and is used to assign a resource group to a node. Once negotiation of the resource group is complete, all nodes in the cluster update their databases and keep track of which node owns the resource group.

In clusters with more than two nodes, the node preference list for each resource group can specify a preferred server plus one or more prioritized alternatives. This enables *cascading failover*, in which a resource group may survive multiple server failures, each time cascading or failing over to the next server on its node preference list. Cluster administrators can set up different node preference lists for each resource group on a server so that, in the event of a server failure, the groups are distributed amongst the cluster's surviving servers.

An alternative to this scheme, commonly called *N+I failover*, sets the node preference lists of all cluster groups. The node preference list identifies the standby cluster nodes to which resources should be moved during the first failover. The standby nodes are servers in the cluster that are mostly idle or whose own workload can be easily pre-empted in the event a failed server's workload must be moved to the standby node.

A key issue for cluster administrators when choosing between cascading failover and N+I failover is the location of the cluster's excess capacity for accommodating the loss of a server. With cascading failover, the assumption is that every other server in the cluster has some excess capacity to absorb a portion of any other failed server's workload. With N+I failover, it is assumed that the "+I" standby servers are the primary location of excess capacity.

Failback

When a node comes back online, the Failover Manager can decide to move some resource groups back to the recovered node. This is referred to as *failback*. The properties of a resource group must have a preferred owner defined in order to failback to a recovered or restarted node. Resource groups for which the recovered or restarted node is the preferred owner will be moved from the current owner to the recovered or restarted node.

Failback properties of a resource group may include the hours of the day during which failback is allowed, plus a limit on the number of times failback is attempted. In this way the cluster service provides protection against failback of resource groups at peak processing times, or to nodes that have not been correctly recovered or restarted.

Global Update Manager

The Global Update Manager (GUM) is used by internal cluster components such as the Failover Manager, Node Manager, or Database Manager in order to replicate changes to the cluster database across cluster nodes in an atomic (either all healthy nodes are updated, or none are updated) and serial (total order is maintained) fashion. GUM updates are typically initiated as a result of a cluster API call. When a GUM update is initiated at a client node, it first requests a “locker” node to obtain a global (where “global” means “across all cluster nodes”) lock. If the lock is not available the client will wait for it.

When the lock becomes available, the locker will grant the lock to the client, and issue the update locally (on the locker node). The client will then issue the update to all other healthy nodes, including itself. If an update succeeds on the locker, but fails on some other node, then that node will be removed from the current cluster membership. If the update fails on the locker node itself, the locker merely returns the failure to the client.

Backup/Restore Manager

The cluster service exposes one API for cluster database backup, BackupClusterDatabase. BackupClusterDatabase contacts the Failover Manager layer first, which then forwards the request to the owner node of the quorum resource. The Database Manager layer in the owner node is then invoked which then makes a backup of the quorum log file and all checkpoint files.

Apart from the API, the cluster service also registers itself at startup as a backup writer with the Volume Shadow Copy Service (VSS). When backup clients invoke the VSS to perform system state backup, it invokes the cluster service to perform the cluster database backup via a series of entry point calls. The server code in the cluster service directly invokes the Failover Manager to perform the backup and the rest of the operation is common with the BackupClusterDatabase API.

The cluster service exposes another API, RestoreClusterDatabase, for restoring the cluster database from a backup path. This API can only be invoked locally from one of the cluster nodes. When this API is invoked, it first stops the cluster service, restores the cluster database from the backup, sets a registry value that contains the backup path, and then starts the cluster service. The cluster service at startup detects that a restore is requested and proceeds to restore the cluster database from the backup path to the quorum resource.

Eventlog Replication Manager

The cluster service interacts with the eventlog service in a cluster to replicate eventlog entries to all cluster nodes. When the cluster service starts up on a node, it invokes a private API in the local eventlog service and requests the eventlog service to bind back to the cluster service. The eventlog service, in response, will bind to the clusapi interface using LRPC. From then on whenever the eventlog service receives an event to be logged, it will log it locally, and then it will drop that event into a persistent batch queue and schedule a timer thread to fire within the next 20 seconds if there is no timer thread active already. When the timer thread fires, it will drain the batch queue and send the events as one consolidated buffer to the cluster service via the cluster API interface to which the eventlog service has bound already.

Once the cluster service receives batched events from the eventlog service, it will drop those events into a local “outgoing” queue and return from the RPC. An event broadcaster thread in the cluster service will drain this queue and send the events in the queue via intracluster RPC to all active remote cluster nodes. The server side API drops the received events into an “incoming” queue. An eventlog writer thread then drains this queue and requests the local eventlog service through a private RPC to write the events locally.

The cluster service uses LRPC to invoke the eventlog private RPC interfaces. The eventlog service also uses LRPC to invoke the cluster API interface for requesting the cluster service to replicate events.

Membership Manager

The Membership Manager (also known as the Regroup Engine) is responsible for maintaining a consistent view of which cluster nodes are currently up or down at a particular moment in time. The heart of the component is a regroup algorithm that is invoked whenever there is evidence that one or more nodes have failed. At the end of the algorithm, all participating nodes will reach identical conclusions on the new cluster membership.

Non-Cluster Service Components

The following components are not considered part of the actual cluster service, but are still closely tied to its operations.

Resource Monitors

Resource Monitors provide the communication interface between resource DLLs and the cluster service. When the cluster service needs to obtain data from a resource, the Resource Monitor receives the request and forwards it to the appropriate resource DLL. Conversely, when a resource DLL needs to report its status or notify the cluster service of an event, the Resource Monitor forwards the information from the resource to the cluster service.

The resource monitor process is spawned as a child process of the cluster service and loads resource DLLs that monitor cluster resources in its process space (loading the resource DLLs in a process separate from the cluster service process helps to isolate faults). Multiple resource monitors can be spawned and executed at the same time. A common property associated with a resource determines if the corresponding DLL is loaded in a separate monitor or if it is loaded in the default monitor. In Windows server 2003 clusters, only one resource DLL can be loaded in a separate monitor, and resource grouping is not allowed. By default, only one resource monitor process is spawned; and all resource DLLs are loaded into this one process.

Each resource monitor functions as an LRPC server for the cluster service process. When the cluster service receives a cluster API call that requires talking to a resource DLL, it will use the LRPC interface to invoke the resource monitor RPC. To receive responses from the resource monitor, the cluster service creates one notification thread per resource monitor process. This notification thread invokes an RPC that is parked permanently in the resource monitor and that thread picks up notifications (such as “resource X is online”) when they are generated. This thread is released only when the resource monitor dies or is stopped explicitly by a shutdown command from the cluster service.

The resource monitor does not maintain any persistent state on its own. All of its initial state is supplied by the cluster service and it merely keeps some limited in-memory state of the resources. The resource monitor communicates with the resource DLLs via well-defined entry points that the DLLs must expose – similar to a COM V-Table. The only operations that the resource monitor does on its own are polling resource DLLs via the “IsAlive” and “LooksAlive” entry points (or alternately checking failure events signaled by resource DLLs), spawning timer threads to monitor pending timeouts of resource DLLs that return ERROR_IO_PENDING from their Online or Offline entry points, and detecting a crash of the cluster service and running down the resources in such a case. The rest of the actions in the resource monitor occur as a result of operations requested by the cluster service via the RPC interface.

The cluster service watches for resource monitor crashes and restarts a monitor if it detects that the process has crashed. No hang detection is done by the cluster service in cluster server today.

The cluster service and resource monitor process share a memory mapped section (backed by the paging file) and the handle to the section is passed to the resource monitor at resource monitor startup. The resource monitor then duplicates the handle. The resource monitor process records the entry point number and resource name into this section just before calling a resource DLL entry point. If the resource monitor crashes, the cluster service (as well as the resource monitor top-level exception filter) reads the shared section to detect the resource and its entry point that caused the resource monitor process to crash.

Event Service

The Event Service serves as the electronic switchboard sending events to and from applications and the cluster service components running on nodes in the cluster. The event processor helps cluster service components disseminate information about important events to all other components and supports the Cluster API eventing mechanism. The Event Processor performs miscellaneous services such as delivering signal events to cluster-aware applications and maintaining cluster objects.

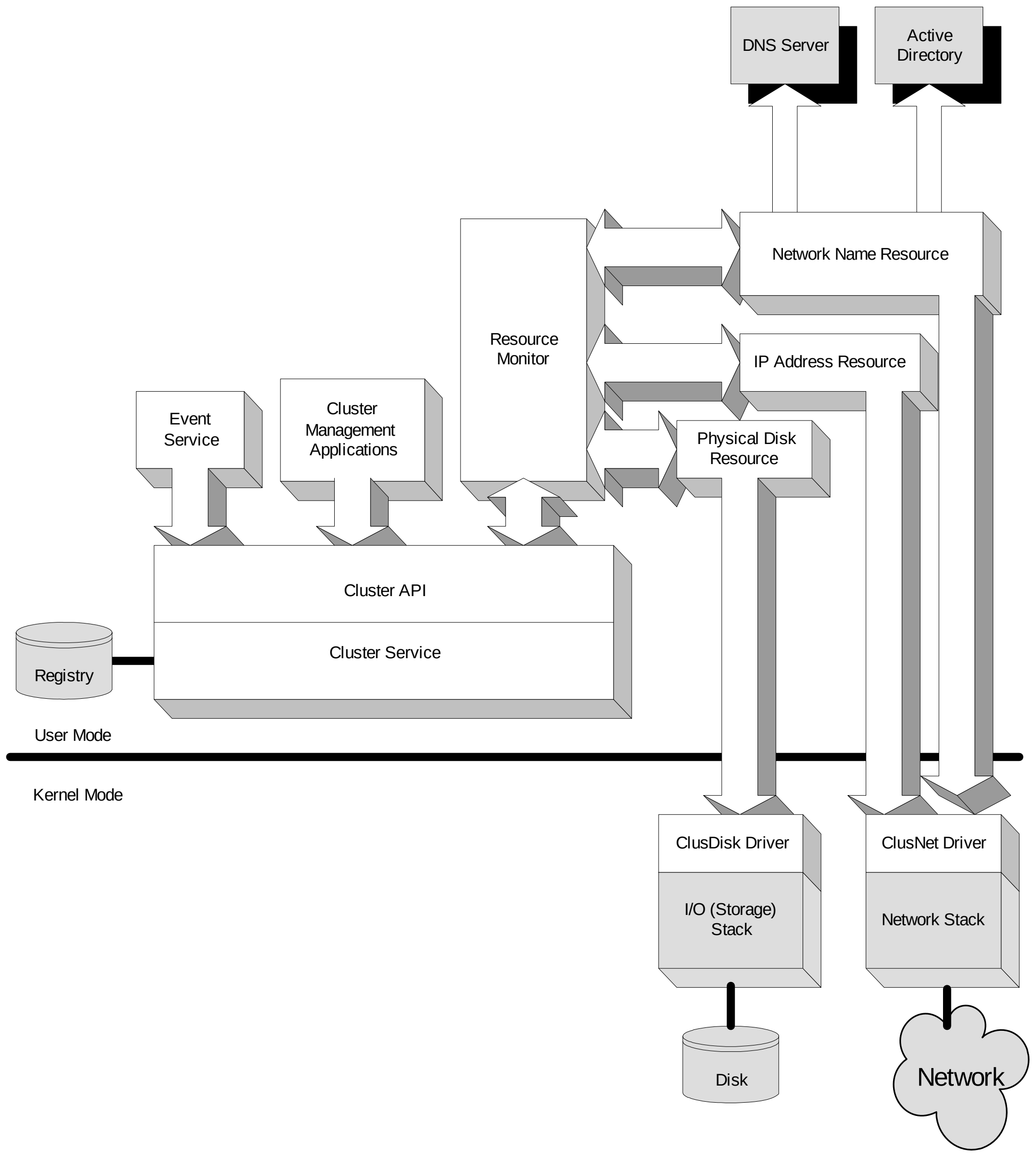

Figure 6 – Server cluster architecture. Non-cluster components are shaded

Cluster Quorum

Each cluster has a special resource known as the *quorum resource*. A quorum resource can be any resource that does the following:

- Provides a means for arbitration leading to membership and cluster state decisions.
- Provides physical storage to store configuration information.

A quorum log is simply a configuration database for the server clustering. It holds cluster configuration information such as which servers are part of the cluster, what resources are installed in the cluster, and what state those resources are in (for example, online or offline). The quorum log is located by default in \MSCS\quolog.log.

There are two main reasons why the quorum is important in a cluster. These are outlined below.

Consistency

Since the basic idea of a cluster is multiple physical servers acting as a single virtual server, it is critical that each of the physical servers have a consistent view of how the cluster is configured. The quorum acts as the definitive repository for all configuration information relating to the cluster. In the event that the Cluster Service is unable to read the quorum log, it will not start, as it is not able to guarantee that the cluster will be in a consistent state, which is one of the primary requirements for a cluster.

Tie-breaker

The quorum is used as the tie-breaker to avoid “split-brain” scenarios. A split-brain scenario happens when all of the network communication links between two or more cluster nodes fail. In these cases, the cluster may be split into two or more partitions that cannot communicate with each other. The quorum is used to guarantee that any cluster resource is only brought online on only one node. It does this by allowing the partition that “owns” the quorum to continue, while the other partitions are evicted from the cluster.

Standard Quorum

As mentioned above, a quorum is simply a configuration database for Microsoft Cluster Service, and is stored in the quorum log file. A standard quorum uses a quorum log file that is located on a disk hosted on a shared storage interconnect that is accessible by all members of the cluster.

Note: It is possible to configure server clusters to use the local hard disk on a server to store the quorum, but this is only supported for testing and development purposes, and should not be used in a production environment.

Each member connects to the shared storage by using some type of interconnect (such as SCSI or Fibre Channel), with the storage consisting of either external hard disks (usually configured as RAID disks), or a storage area network (SAN), where logical slices of the SAN are presented as physical disks.

Note: It is important that the quorum uses a physical disk resource, as opposed to a disk partition, as the entire physical disk resource is moved during failover.

Standard quorums are available in Windows NT 4.0, Enterprise Edition, Windows 2000, Advanced Server, Windows 2000, Datacenter Edition, Windows Server 2003, Enterprise Edition, and Windows Server 2003, Datacenter Edition, and are illustrated in Figure 7.

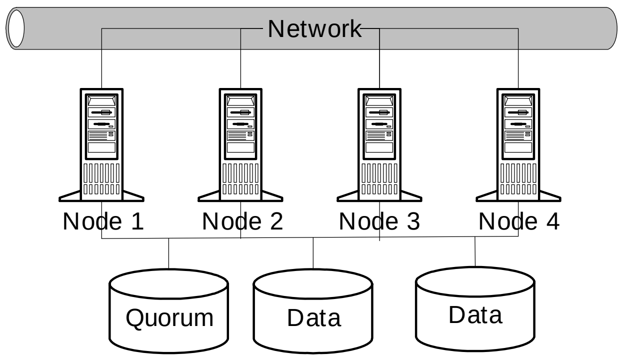

Figure 7 -- Diagram of a standard quorum in a four-node cluster

Majority Node Set Quorums

A majority node set (MNS) quorum is a single quorum resource from a server cluster perspective. However, the data is actually stored by default on the system disk of each member of the cluster. The MNS resource takes care to ensure that the cluster configuration data stored on the MNS is kept consistent across the different disks. Figure 8 depicts a four-node cluster with an MNS quorum configuration.

Majority node set quorums are available in Windows Server 2003 Enterprise Edition, and Windows Server 2003 Datacenter Edition.

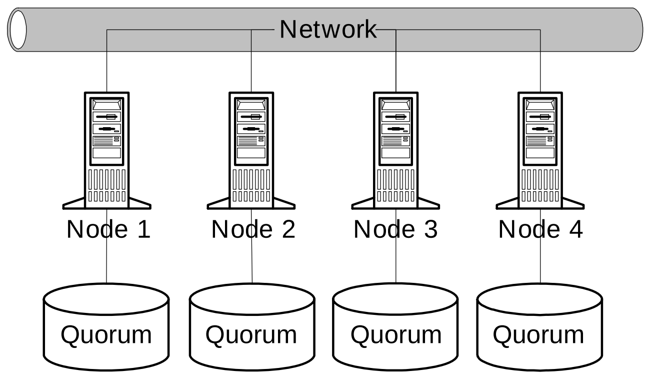

Figure 8 -- Diagram of an MNS quorum in a four-node cluster

While the disks that make up the MNS could, in theory, be disks on a shared storage fabric, the MNS implementation that is provided as part of Windows Server 2003 uses a directory on each node’s local system disk to store the quorum data. If the configuration of the cluster changes, that change is reflected across the different disks. The change is only considered to have been committed, i.e. made persistent, if that change is made to:

(<Number of nodes configured in the cluster>/2) + 1

This ensures that a majority of the nodes have an up-to-date copy of the data. The cluster service itself will only start up, and therefore bring resources online, if a majority of the nodes configured as part of the cluster are up and running the cluster service. If there are fewer nodes, the cluster is said not to have quorum and therefore the cluster service waits (trying to restart) until more nodes try to join. Only when a majority or quorum of nodes are available, will the cluster service start up, and the resources be brought online. In this way, since the up-to-date configuration is written to a majority of the nodes regardless of node failures, the cluster will always guarantee that it starts up with the latest and most up-to-date configuration.

In the case of a failure or split-brain, all partitions that do not contain a majority of nodes are terminated. This ensures that if there is a partition running that contains a majority of the nodes, it can safely start up any resources that are not running on that partition, safe in the knowledge that it can be the only partition in the cluster that is running resources (since all other partitions are terminated).

Given the differences in the way the shared disk quorum clusters behave compared to MNS quorum clusters, care must be taken when deciding which model to choose. For example, if you only have two nodes in your cluster, the MNS model is not recommended, as failure of one node will lead to failure of the entire cluster, since a majority of nodes is impossible.

Cluster Resources

The cluster service manages all resources as identical opaque objects by using Resource Monitors and resource DLLs. The Resource Monitor interface provides a standard communication interface that enables the cluster service to initiate resource management commands and obtain resource status data. Actual command execution and data is obtained by the Resource Monitor through the resource DLLs. The cluster service uses resource DLLs to bring resources online, manage their interaction with other resources in the cluster, and—most importantly—to monitor their health to detect failure conditions.

Server clusters provide resource DLLs to support both Microsoft cluster-aware applications and generic non-cluster-aware applications from independent software vendors (ISVs) and third-party companies. Additionally, ISVs and third parties can provide resource DLLs that make their specific products cluster-aware. For more information about available cluster-aware applications and hardware, see the section “For More Information.”

To enable resource management, a resource DLL needs only to expose a few simple resource interfaces and properties. Resource Monitor loads a particular resource DLL into its address space as privileged code running under the system account. The system account is an account used only by the operating system and services integrated with the base operating system. Using the system account enables the cluster service to perform various functions within the context of the operating system. For more information about the architecture of Windows Server 2003 or Windows 2000 system services, and account security, see:

http://www.microsoft.com/windows2000/techinfo/howitworks/default.asp

All resource DLLs provided by Microsoft for Microsoft cluster-aware applications run in a single Resource Monitor process. ISV or third-party resource DLLs will require their own resource monitor. Resource monitors are created by the cluster service as needed when a resource is installed or started on a cluster node.

When resources depend on the availability of other resources to function, these dependencies can be defined by the resource DLL. In the case where a resource is dependent on other resources, the cluster service will bring it online only after the resources on which it depends are brought online in the correct sequence.

Resources are taken offline in a similar manner. The cluster service takes resources offline only after any dependent resources have been taken offline. This prevents introducing circular dependencies when loading resources.

Each resource DLL can also define the type of computer and device connection that is needed by the resource. For example, a disk resource may require ownership only by a node that is physically connected to the disk device. Local restart policies and desired actions during failover events can also be defined in the resource DLL.

Resource DLLs provided with Windows server 2003 products enable server clusters to support the following resources:

- File and print shares
- Generic services or applications

- Generic scripts
- Physical disks
- Microsoft Distributed Transaction Coordinator (MSDTC)
- Internet Information Services (IIS)
- Message Queuing (MSMQ) Triggers
- Network addresses and names

Windows Server 2003, Enterprise Edition, and Windows Server 2003, Datacenter Edition, include resource DLLs for the following additional services:

- Distributed File System (DFS)
- Dynamic Host Configuration Protocol (DHCP) service
- Windows Internet Service (WINS)

Cluster-aware applications that provide their own resource DLLS and resource monitors enable advanced scalability and failover benefits. For example, a database server application with its own database resource DLL enables the cluster service to fail over an individual database from one node to another. Without the unique database resource DLL, the database application would be run on the cluster using the default generic server application resource DLL. When using the generic application server resource DLL, the cluster service can only fail over an entire generic server application (and all its databases). Individual resource DLLs, however, such as the example database resource DLL, enable treating the database as a resource that can be monitored and managed by the cluster service. Thus, the application is no longer the only resource and failover unit that can be managed by server clusters. This enables simultaneously running multiple instances of the application on different nodes in the cluster, each with its own set of databases. Providing resource DLLs that define application-specific resources is the first step towards achieving a cluster-aware application.

For information about creating a Cluster Server resource DLL, see:
http://msdn.microsoft.com/library/backgrnd/html/msdn_mscs_resource_dlls.htm

Cluster Administration

A cluster is managed using the Cluster Administrator, a graphical administrator's tool that enables performing maintenance, monitoring, and failover administration. Additionally, server clusters provide an automation interface that can be used to create custom scripting tools for administering cluster resources, nodes, and the cluster itself. Applications and administration tools, such as the Cluster Administrator, can access this interface using RPC regardless of whether the tool is running on a node in the cluster or on an external computer. The administrative interface provides access to the cluster component managers described in this document to enable management of cluster entities such as nodes, resources, resource groups, and the cluster itself. For information about developing an administration tool using the automation interface, see the Windows Clustering section of the Platform Software Developer Kit (SDK):

http://www.msdn.microsoft.com/library

For information about using Cluster Administrator, see the product help in Windows Server 2003 Enterprise Edition, Windows Server 2003 Datacenter.

Cluster Formation and Operation

When the cluster service is installed and running on a server, the server is available to participate in a cluster. Cluster operations will reduce single points of failure and enable high availability of clustered resources. The following sections briefly describe node behavior during cluster creation and operation.

Note: For information about installing Cluster Server, see the Windows server 2003 product family help and deployment guides.

Creating a Cluster

Server clusters include a cluster installation utility to install the cluster software on a server and to create a new cluster. To create a new cluster, the utility is run on the computer selected to be the first member of the cluster. This first step defines the new cluster by establishing a cluster name and creating the cluster database and initial cluster membership list. New to Windows server 2003 clusters is a cluster administration setup wizard, plus the ability to create a cluster, even remotely, using the cluster.exe command line interface.

The next step towards creating a cluster is adding the common data storage devices that will be available to all members of the cluster. This establishes the new cluster with a single node and with its own local data storage devices and the cluster common resources—generally disk or data storage and connection media resources.

The final step in creating a cluster is running the installation utility on each additional computer that will be a member in the cluster. As each new node is added to the cluster, it automatically receives a copy of the existing cluster database from the original member of the cluster. When a node joins or forms a cluster, the cluster service updates the node's private copy of the configuration database.

Forming a Cluster

A server can form a cluster if it is running the cluster service and cannot locate other nodes in the cluster. To form the cluster, a node must be able to acquire exclusive ownership of the quorum resource.

When a cluster is initially formed, the first node in the cluster contains the cluster configuration database. As each additional node joins the cluster, it receives and maintains its own local copy of the cluster configuration database. The quorum resource stores the most current version of the configuration database in the form of recovery logs that contain node-independent cluster configuration and state data.

During cluster operations, the cluster service uses the quorum recovery logs to perform the following:

- Guarantee that only one set of active, communicating nodes is allowed to form a cluster
- Enable a node to form a cluster only if it can gain control of the quorum resource
- Allow a node to join or remain in an existing cluster only if it can communicate with the node that controls the quorum resource

From the point of view of other nodes in the cluster and the cluster service management interfaces, when a cluster is formed, each node in the cluster may be in one of three distinct states. These states are recorded by the Event Processor and replicated by the Eventlog Manager to other clusters in the node. Cluster service states are:

- **Offline**. The node is not a fully active member of the cluster. The node and its cluster server may or may not be running.
- **Online**. The node is a fully active member of the cluster. It honors cluster database updates, contributes votes to the quorum algorithm, maintains heartbeats, and can own and run resource groups.
- **Paused**. The node is a fully active member of the cluster. It honors cluster database updates, contributes votes to the quorum algorithm, and maintains heartbeats, but it cannot accept resource groups. It can only support those resources groups for which it currently has ownership. The paused state is provided to allow certain maintenance to be performed. Online and paused are treated as equivalent states by the majority of the server cluster components.

Joining a Cluster

To join an existing cluster, a server must be running the cluster service and must successfully locate another node in the cluster. After finding another cluster node, the joining server must be authenticated for membership in the cluster and receive a replicated copy of the cluster configuration database.

The process of joining an existing cluster begins when the Windows Server 2003 or Windows 2000 Service Control Manager starts the cluster service on the node. During the start-up process, the cluster service configures and mounts the node's local data devices. It does not attempt to bring the common cluster data devices online as nodes because the existing cluster may be using the devices.

To locate other nodes, a discovery process is started. When the node discovers any member of the cluster, it performs an authentication sequence. The first cluster member authenticates the newcomer and returns a status of success if the new server is successfully authenticated. If authentication is not successful, in the case where a joining node is not recognized as a cluster member, or has an invalid account password, the request to join the cluster is refused.

After successful authentication, the first node online in the cluster checks the copy of the configuration database on the joining node. If it is out-of-date, the cluster node that is authenticating the joining server sends it an updated copy of the database. After receiving the replicated database, the node joining the cluster can use it to find shared resources and bring them online as needed.

Leaving a Cluster

A node can leave a cluster when it shuts down or when the cluster service is stopped. However, a node can also be forced to leave (evicted) when the node fails to perform cluster operations, such as failure to commit an update to the cluster configuration database.

When a node leaves a cluster in the event of a planned shutdown, it sends a **ClusterExit** message to all other members in the cluster, notifying them that it is leaving. The node does not wait for any responses and immediately proceeds to shut down resources and close all cluster connections.

Because the remaining nodes received the exit message, they do not perform the same regroup process to reestablish cluster membership that occurs when a node unexpectedly fails or network communications stop.

Failure Detection

Failure detection and prevention are key benefits provided by server clusters. When a node or application in a cluster fails, server clusters can respond by restarting the failed application or dispersing the work from the failed system to surviving nodes in the cluster. Server cluster failure detection and prevention includes bi-directional failover, application failover, parallel recovery, and automatic failback.

The cluster service detects failures of individual resources or an entire node and dynamically moves and restarts application, data, and file resources on an available, healthy server in the cluster. This allows resources such as database, file shares, and applications to remain highly available to users and client applications.

Server clusters are designed with two different failure detection mechanisms:

- **Heartbeat** for detecting node failures.
- **Resource Monitor and resource DLLs** for detecting resource failures.

Detecting Node Failures

Periodically, each node exchanges datagram messages with other nodes in the cluster using the private cluster network. These messages are referred to as the *heartbeat*. The heartbeat exchange enables each node to check the availability of other nodes and their applications. If a server fails to respond to a heartbeat exchange, the surviving servers initiate failover processes including ownership arbitration for resources and applications owned by the failed server. Arbitration is performed using a challenge and defense protocol. In other words, the node that appears to have failed is given a time window where it is expected to demonstrate in any of several ways that it is still up and running properly, and can communicate with the surviving nodes. If it is unable to do that, only then is it removed from the cluster.

Failure to respond to a heartbeat message can be caused by several events, such as computer failure, network interface failure, network failure, or even periods of unusually high activity. Normally, when all nodes are communicating, Configuration Database Manager sends global configuration database updates to each node. However, when a failure on a heartbeat exchange occurs, the Log Manager also saves configuration database changes to the quorum resource. This ensures that surviving nodes can access the most recent cluster configuration and local node registry key data during recovery processes.

Note that the failure detection algorithm is very conservative: in other words, it gives the apparently failed node the benefit of the doubt as much as possible before proceeding with the failover process. Clearly, if the cause of the heartbeat response failure is temporary, it is best to avoid the potential disruption a failover might cause. However, there is no way to know whether the node is going to be quiet for another millisecond, another second, or, on the other end of the scale, it has suffered a catastrophic failure. As a result, after a “reasonable” period of time, a failover is initiated.

Detecting Resource Failures

The Failover Manager and Resource Monitors work together to detect and recover from resource failures. Resource Monitors keep track of resource status by periodically polling resources using the resource DLLs. Polling involves two steps, a cursory *LooksAlive* query and longer, more definitive, *IsAlive* query. When the Resource Monitor detects a resource failure, it notifies the Failover Manager and continues to monitor the resource.

The Failover Manager maintains resources and resource group status. It is also responsible for performing recovery when a resource fails and will invoke Resource Monitors in response to user actions or failures.

After a resource failure is detected the Failover Manager can perform recovery actions that include either restarting a resource and its dependent resources, or moving the entire resource group to another node. Which recovery action is performed is determined by resource and resource group properties and node availability.

During failover, the resource group is treated as the failover unit, to ensure that resource dependencies are correctly recovered. Once a resource recovers from a failure, the Resource Monitor notifies the Failover Manager, which can then perform automatic failback of the resource group, based on the configuration of the resource group failback properties.

Future Directions

As Windows–based products evolve, the future development of server clusters will focus on the following key areas:

- Easier installation and verification of cluster configurations, including support for new types of hardware.
- Continued support for geographically dispersed clusters, enabling disaster-tolerant configurations.
- Simpler, more powerful management of cluster-based applications and services, including continued focus on scripted, remote, and “lights out” management.
- Extension of cluster-based availability and scalability benefits to even more system services.
- Tighter integration of the infrastructure and interfaces of all Windows–based clustering technologies to enhance performance, flexibility, and manageability.
- Continued support for third-party ISVs and corporate developers to simplify the development, installation, and support of cluster-aware applications, both for higher availability and for higher scalability.

Note: Third-party developers can create unique, cluster-aware quorum resource types that meet the above requirements. For information about developing cluster-aware products, visit the Platform SDK Web site on MSDN online:
http://www.msdn.microsoft.com/library

For More Information

Books

Windows NT Microsoft Cluster Server, by Richard R. Lee, Osborne McGraw-Hill, 1999.

Windows 2000 Cluster Server Guidebook, by David Libertone, Prentice Hall, 2000.

Windows NT Backup & Recovery, by John McMains and Bob Chronister, Osborne McGraw-Hill, 1998.

Windows NT Clustering Blueprints, by Mark A. Sportack, SAMS Publishing, 1997.

In Search of Clusters, Second Edition: The Coming Battle in Lowly Parallel Computing, Gregory F. Pfister, Prentice Hall, 1998, ISBN: 0138997098.

The Book of SCSI, Peter M. Ridge, No Starch Press, 1995, ISBN: 1886411026.

Transaction Processing Concepts and Techniques, Gray, J., Reuter A., Morgan Kaufmann, 1994. ISBN 1558601902, survey of outages, transaction techniques.

Related Links

You can also visit the Microsoft Web site to learn more about any of the Windows clustering technologies.

For information about the Windows 2000 Server family of products, see:

http://www.microsoft.com/windows2000/server/default.asp

For information about Windows 2000 Server Reliability & Availability, see:
http://www.microsoft.com/windows2000/server/evaluation/business/overview/reliable/default.asp

For information about Windows 2000 Clustering Technologies, see:
http://www.microsoft.com/windows2000/technologies/clustering/default.asp

For information about what's new in Windows server 2003 clusters, see:
http://www.microsoft.com/windows.netserver/evaluation/overview/technologies/clustering.asp

For documentation about the architecture of Windows base services, including Windows Clustering technologies, see *Windows Base Services* under *Windows Development* in the Platform Software Developer Kit (SDK):

http://www.msdn.microsoft.com/library