Architectural Design:
A Scalable, Highly Available Business Object Architecture

White Paper

Abstract

Building a scalable, highly available application is a necessary process in becoming functional as an e-commerce entity. This white paper introduces a Microsoft® Windows® DNA-based workflow architecture that can help to facilitate the challenging design process. The architecture scales easily across multiple high-availability servers.

Microsoft Corporation • One Microsoft Way • Redmond, WA 98052-6399 • USA

0400

Contents

Introduction **1**

Designing for Scalability **2**

Multitier Architecture 2

Messaging and Scalability 3

A Scalable Workflow Architecture 3

Replication and Scalability 5

Distributed Database 7

Designing for Availability **9**

Failover Clustering 9

An Available Workflow Architecture 10

Messaging and Availability 10

Replication and Availability 10

Business Application Integration **12**

Summary **14**

For More Information 14

Introduction

Successful commerce Web sites are experiencing ten-fold and even one hundred-fold increases in order volume as more and more business moves to the Internet. It is clear that a scalable order fulfillment process that remains ready to serve at any time and never fails is a requirement.

To fully understand the necessity of building scalable, highly available applications, consider some scenarios where such applications are useful. For example, a manufacturer that supplies computer parts in a *just-in-time* fashion must have a fully reliable parts-ordering process. Any failure might cause the entire shipping line to shut down, costing millions of dollars. Or, consider an online shoe seller that serves customers in many different parts of the world. Customers might order merchandise at any time of the day or night and on any day of the week. If the order fulfillment application is unavailable for even an hour, the result could be customer and revenue loss. In both examples, the business applications need to be scalable and available twenty-four hours a day, seven days a week.

This paper reviews some standard scalability and availability fundamentals. It also introduces a real-world workflow architecture that can be used by your organization to create a scalable, high availability application running on multiple servers. You will learn how to combine Microsoft technologies to create the most powerful enterprise solution available today.

Designing for Scalability

Multitier Architecture

The multitier architecture is a fundamental requirement for building a scalable application. Instead of designing an application to run solely on one computer, it is designed to run on multiple tiers of computers. This is a principle upon which Windows DNA is built. For additional information on the Windows DNA platform, see “For More Information.”

A multitier application is constructed across multiple logical or physical tiers. Historically, a two-tier architecture, represented by the division between client-based work and server-based work was most common. Eventually, the three-tier architecture was constructed, moving business logic into a middle tier, sometimes placing it on a separate application server (see Figure 1). With the prevalence of the Internet, a simple browser allowed all of the user interface logic to be moved to the Web server. This is sometimes referred to as a 3½-tier architecture.

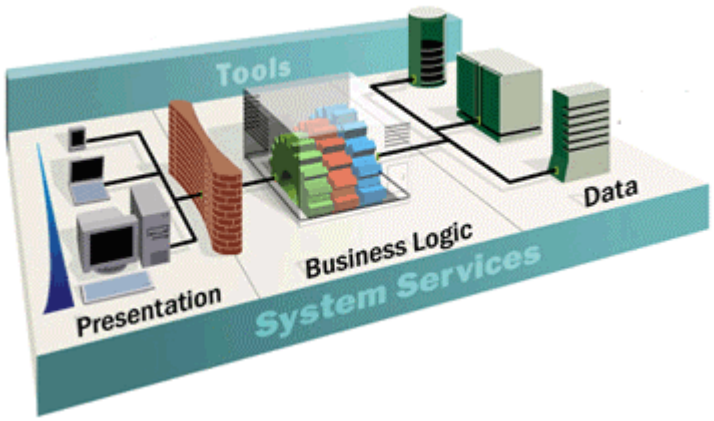

Figure 1: A multitier architecture

With a Windows DNA-based application, the client application might be as simple as a browser running on a remote workstation. The server side application is an Active Server Pages (ASP) or Internet Server Application Programming Interface (ISAPI) application hosted by a Web server, which may be executing scripts. The scripts invoke objects that execute business logic. The business objects then manipulate data located on a database server.

All of the server-side operations can occur on one or multiple servers. The multitier approach offers the benefit of distributing loads across multiple servers, providing for more scalability than that offered through a single server solution.

An important part of Windows DNA, the Component Object Model (COM) allows a developer to encapsulate business functionality. And, because components represent discrete tasks, they are also easier to develop, test, and reuse. Applying this to the Web application architecture described above, the

scripts executed on the Web server invoke COM objects that implement the business logic.

COM objects lend themselves to building scalable applications in three ways. First, they remove excess business logic from the presentation and data tiers, dedicating each tier to specific tasks. Second, COM objects can share resources, such as data connections. Third, COM objects can be distributed to multiple servers. COM objects also promote high availability of applications by allowing multiple servers to perform the same task.

Messaging and Scalability

Another building block for constructing a scalable application is asynchronous messaging. When incoming workloads cannot be predicted, or steps within the application take varying amounts of time to complete, many distributed applications require the ability to handle delays between a request and a response. Message Queuing allows applications to use components that communicate with one another using queued messages and ensures that messages are routed securely and robustly to and from message queues.

Being able to write applications that do not require immediate responses from either clients or servers allows developers to provide the flexibility required to handle real-world conditions, such as routine pauses within business processes, the onslaught of peak traffic conditions, and temporary network outages (depending on the physical distribution of servers). Using Message Queuing also helps developers write applications that are more readily available and scalable. For additional information on Message Queuing, see “For More Information.”

When developers write components that fully take advantage of Message Queuing, their application can send messages to another application without waiting for a response. Because the receiving application might be processing other tasks, those messages are sent to a queue, where they are stored until the receiving application is available to remove them.

A Scalable Workflow Architecture

Fulfilling an e-commerce order involves multiple steps. For example, these steps might include preparing the order, authorizing a credit card, shopping for a supplier, and shipping the requested product.

COM components and Message Queuing can be used to process each of these steps, as shown in Figure 2. Each component receives work from its inbound queue, completes its processing task, and then sends a message to the queue of the next component. In cases where steps require more processing and, as a result, take longer than others, orders can be dispatched to multiple order processors so that work can be performed in parallel. Orders

can also be dispatched to components running on other computers, expanding the resources applied to processing-intensive steps as much as necessary.

Figure 2 shows scalable workflow architecture. A Drainer removes a message from a queue and invokes a Processor, which is an individual COM object. The architecture scales by virtue of the ability to dispatch work to multiple queues and multiple computers.

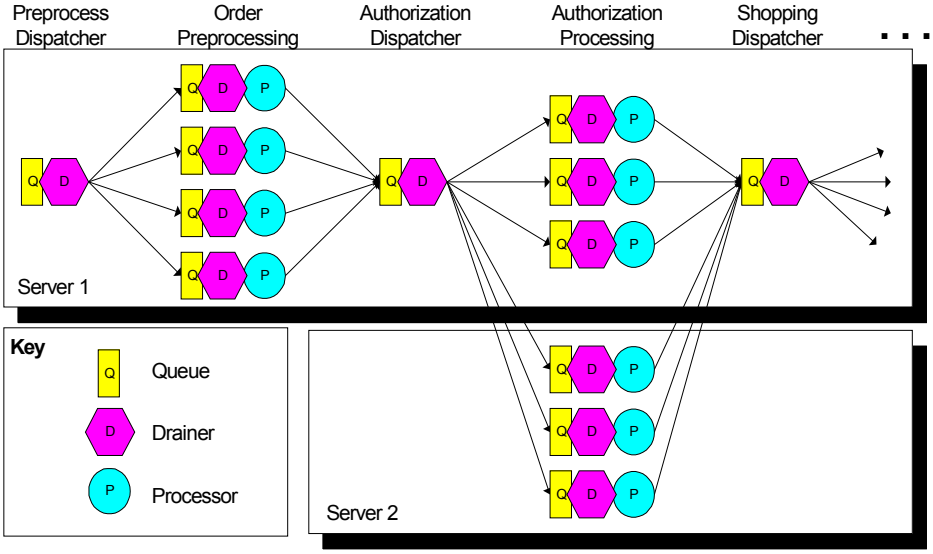

Figure 2: Scalable workflow architecture

Consider the flow of an order through the system. An order is first stored in the remote database (not shown) and the corresponding record key, in this case the order identifier (ID), is placed in the preprocess dispatch queue. A small program called a queue drainer checks the preprocess dispatch queue for messages. If one is present, it retrieves the message and then places it, in round-robin order, in one of the four order-preprocessing queues.

Each of the order-preprocessing queues has its own queue drainer that awaits the arrival of messages. The queue drainer retrieves a message and invokes a COM component to process the order through this step. While an order-preprocessing component is busy, should other messages arrive in the queue, they will be stored until the component is finished. Further discussion of the importance of queue drainers is covered in “Designing for Availability”. When the queue drainer invokes the order-preprocessing COM component, it passes the order ID retrieved from the message. Next, the component retrieves the order from the database and applies the business logic required to complete this step. This might include updating the order record, or inserting or updating data in other related database tables. The final step of the process will be the posting of a message to the next stage, the authorization dispatch queue.

Assuming the authorization step takes about 50 percent longer to process as the order-preprocessing step, the authorization dispatcher disperses messages to six authorization-processing queues. For illustration purposes, three of the

authorization processors are running on a different server. Additional servers can be employed if further scaling is required. After the authorization-processing step, an order continues in a similar fashion through the remaining steps in the order processing workflow.

Some Technical Details

It might be argued that the extra data retrieval calls at each step in the workflow represent a potential bottleneck. Because the order processing steps are broken out into smaller units of work and each step separately retrieves and updates data, it is natural to assume that more database accesses could lead to a performance problem. The tradeoff for scalability here assumes that most of the processing time does not occur on the database tier, but instead on the business logic (application server) tier. In other words, the database is used more for what it does best: to store and retrieve data. If business logic is placed at the database tier, then there is a much higher chance for performance problems. It is also important to note that the only data passed from process to process through the queue is the order ID. A state management record is not required because the state of any order is implied by the location of its ID within the workflow. A database table is checked by each dispatcher and COM component to determine the next queue to which a message is to be posted.

The goal of this architecture is to have the flexibility to distribute the processing load to additional application-tier servers. The dispatcher routing technique allows you to process the messages on different servers. To add a new server, you install the necessary COM objects and then update the dispatch control records. A well-implemented dispatcher might reread these control records periodically and, as the result, dynamically allocate work to the new server.

The load-balancing algorithm at each dispatcher can be a simple round-robin algorithm, causing all processing components to receive an equal number of messages. You can use an alternative algorithm, such as one that requests the number of items in each of the target queues. The message could then be placed in the shortest queue.

Replication and Scalability

Another building block for constructing a scalable application is database replication. Replication is the duplication of data from a source database to a destination database, usually on separate servers. Essentially, part of a database is reproduced on another server. For additional information on SQL Server replication, see “For More Information.”

To demonstrate scaling with replication, consider the real-world activity of looking up a customer record. A typical query sequence might be:

1. Retrieve all customers with the last name of Dunn.

2. Of the results list in 1, retrieve those with the first name of Matthew.

3. Of the results list in 2, retrieve the order numbers and order dates.

4. Of the results list in 3, retrieve the order status for a specific order number.

If hundreds of these sequences are executing at the same time, which is often the case in a customer call center, this sequence would not be a very efficient use of computing resources—especially on a database server that is filling multiple orders. If, however, the customer table and order cross-reference table were replicated to another server, the first three retrievals would operate independently of the main database. Once the customer and corresponding order number is located, a simple, efficient retrieval can be made to the main database for the order status.

You can apply replication to enhance the scalability of the e-commerce order fulfillment example. One of the functions of order fulfillment might be to ship items directly from a remote vendor. Assuming that the vendor has a similar order fulfillment system, an order on the central server is replicated to a server at the vendor site. When a replicated order is inserted into the order table at the vendor site, an insert trigger can be used to post a message to the initial queue of the vendor site order-processing workflow (note that the trigger must call an extended stored procedure that, in turn, can then post the message using the Message Queuing API.) Once the message is posted, the vendor server fulfills the order through a series of steps; ultimately replicating the results back to the central site.

SQL Server replication itself is scalable. Consider the roles of publisher, distributor, and subscriber in determining how replication works. A Publisher is a server that makes data available for replication to other servers. Subscribers are servers that store replicated data and receive updates. The Distributor is a server that does the additional work of storing and copying data to multiple subscribers.

Figure 3 shows a scalable replication model in which data is first copied from Publisher to Distributor and then from the Distributor to all Subscribers.

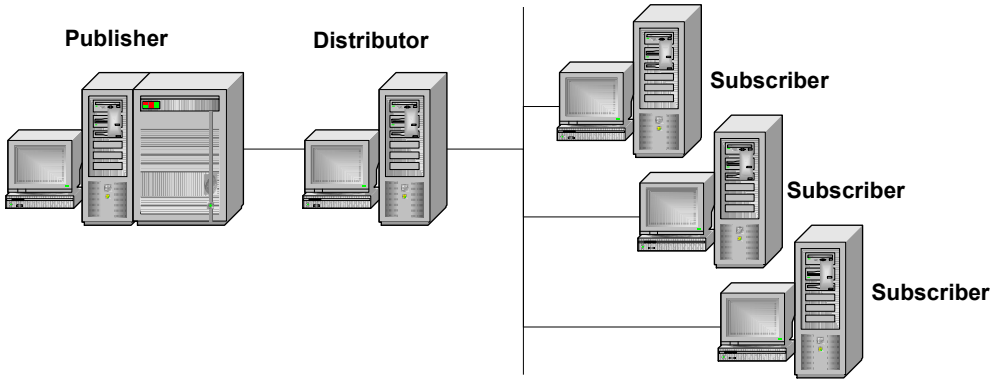

Figure 3: The scalable replication model

Distributed Database

An additional building block for constructing a scalable application is a distributed database, which differs from replication. A distributed database is implemented on a network in which the component partitions are distributed over various network nodes. Depending on the specific update and retrieval traffic, distributing the database can enhance overall performance significantly.

A distributed database decentralizes the main database across multiple servers. The workload against certain tables can be isolated to one server. A carefully designed distributed database can have simultaneous operations being performed on different areas of the database with no mutual performance impact.

Going back to the example of looking up a customer record, consider the two separate activities of order processing and customer/order status lookup. To distribute the database effectively, the customer table could be placed on one server, while the order table could be placed on another. The third table, the customer/order cross-reference table, is located with the customer table since this data is helpful to customer service and is not useful to order fulfillment. With this arrangement, order processing is not impaired by customer service. Only two efficient communications are necessary between the servers, occurring when:

1. An order is created (an insertion is made to the customer/order cross-reference table).
2. Order status is retrieved for a specific order.

An alternative method of distributing the database is to split the order table across multiple databases. For example, order IDs might be generated with a prefix to indicate the server on which they are stored. During order processing, the COM object can examine the ID and determine which server to access to retrieve an order.

Microsoft SQL™ Server 7.0 supports distributed queries between multiple SQL Server 7.0 servers as well as to any OLE DB provider.

Designing for Availability

Failover Clustering

In an e-commerce order fulfillment center, a server failure can be very costly when order processing and product shipping are a 24X7 operation. Microsoft's primary availability solution is a failover clustering solution called Windows Clustering, available with Windows 2000 Advanced Server. For additional information on Windows Clustering, see "For More Information."

Failover clustering provides a server with a backup partner in case of emergency. If one system fails, the partner takes over. As a result, the user of a robustly written application does not notice any functional failure. The only thing that the user may notice is that the task took a bit longer than usual. Within the application, a failover requires that database connections be reinstated and transactions be restarted.

With Windows NT 4.0 Enterprise Edition, an active/passive solution was available, where one node of a pair runs actively, while the other waits passively for a failover. To reduce the expense of having one of the two nodes sitting idle, Windows 2000 introduces a more efficient active/active solution where both nodes can run active processes such as Message Queuing or SQL Server.

Figure 4 shows a sample server cluster configuration, demonstrated by LAN and shared disk array connectivity in a two-node cluster. If a hardware failure occurs on Node 1, Node 2 takes over.

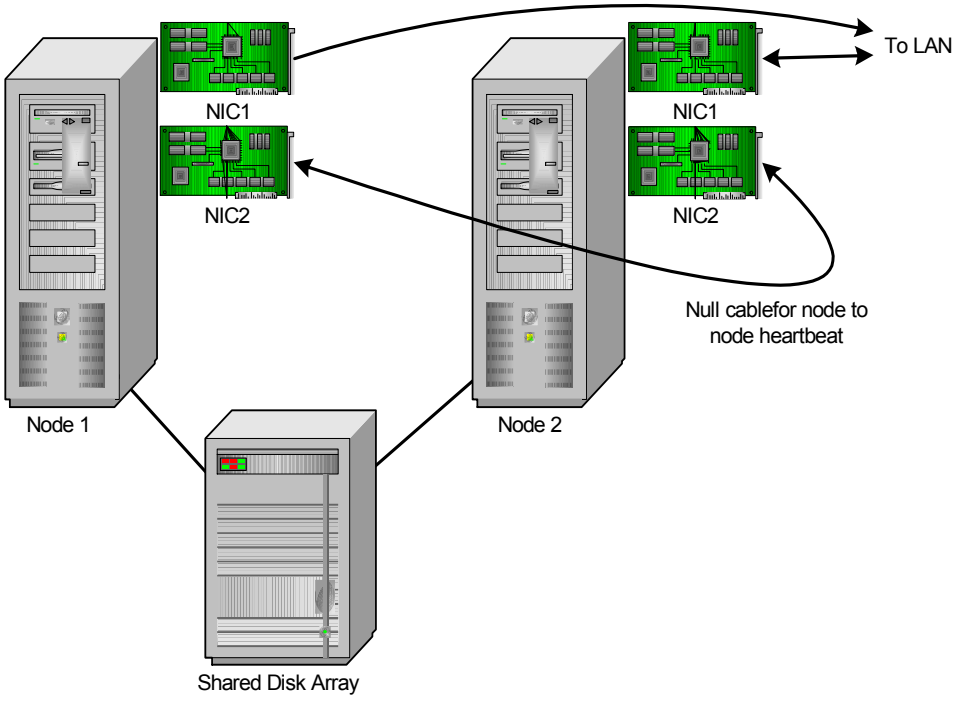

Figure 4: A sample server cluster configuration

In Figure 4, a sample cluster layout includes two server nodes physically wired to share a common disk array. Each node contains two network interface cards (NICs). The extra NIC is used for private communication (heartbeat) between the nodes, monitoring system health. Each node owns resources, such as hard disks, SQL Server, or Message Queuing. To illustrate, assume that Node 1 owns disk D and SQL Server 1, and Node 2 owns disk E and SQL Server 2. In the event of failure on Node 1, the resources of that node fail over to Node 2. In this case, Node 2 becomes the new owner of disk D and SQL Server 1. Thus, when one node in a cluster fails, the cluster resources are moved to the other node.

An Available Workflow Architecture

Consider the practical application of failover clustering to the scalable workflow architecture of the preceding section. As previously mentioned, the queue drainer reads messages from a queue and either dispatches them to other queues or invokes COM objects to execute specific business logic.

Each queue drainer operates as a generic application resource in the cluster. These resources are made dependent on Message Queuing. If a computer failure occurs on one of the nodes, a failover event occurs and the cluster responds by restarting Message Queuing and the queue drainers on the alternate node. The queue drainers simply continue by checking the queue and invoking COM objects.

It is important to use transactional queues so that, in the event of failure, partially processed messages are rolled back. For example, suppose a transaction was started, a message was read from a queue, and a COM object was invoked. Then, in the middle of processing some business logic, a failure occurs. As a result, the entire transaction is rolled back. When the queue drainer is restarted on the alternate node, the same message is reread and the transaction is restarted.

Messaging and Availability

Message Queuing can also be used to simulate increased network availability by providing store and forward functionality. Messages generated by processes on a local system that are destined for a remote system are stored on the local system if the remote system is unavailable. Processes on the local system can continue to operate and post messages to the queue. When the remote system becomes available, the queued messages are then processed.

Replication and Availability

Using the database replication service can also improve system availability. Database replication services permit an operational publishing database to be

updated while a subscriber database is unavailable. Changes to the operational database are queued and applied to the subscriber database when it becomes available.

Business Application Integration

Figure 5 shows the integration of all components into one large-scale operation. Each of the systems in the diagram is made up of two clusters—an application server cluster and a database server cluster. Each cluster consists of two multi-processor servers sharing a common disk drive array (physically wired as demonstrated in Figure 4).

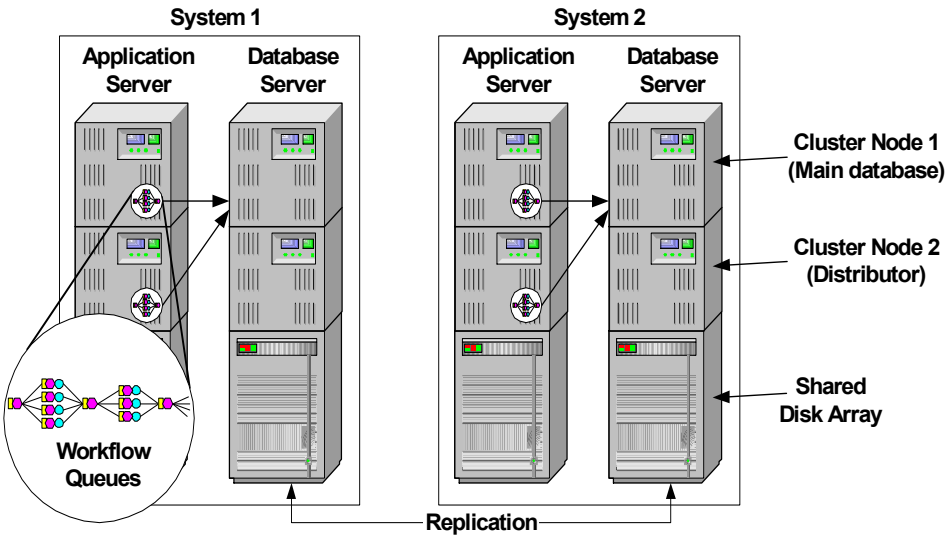

Figure 5: A sample large-scale operation configuration

Message Queuing runs on the application servers and is configured as a cluster resource to ensure availability. Also running on the application tier are a number of COM objects that serve as order processors. These COM objects communicate with one of the database nodes in the cluster. The alternate database node in the cluster serves as the replication distribution server.

In Figure 6, on the following page, orders that originate from the Web site are transferred into the database on System 1 through replication. When the orders are inserted into the database, a message is placed on the initial order-processing dispatch queue in the application server tier. From there, the workflow process takes over and processes the order through its various stages. For example, in some organizations, orders originate from several sources. A first stage might be to reorganize these orders into a common format. Once the order is in a common format, it can move to the next stage in the process that determines how to fulfill the order. At this stage, inventories might be checked on various systems throughout the network and shipment costs assessed. The next stage might be to transfer the order to a remote System 2 for further processing and fulfillment.

The transfer can occur in a manner similar to the method by which the order was transferred into System 1. The order is first replicated to the remote system and, upon insertion, a message is placed on the dispatch queue to start the workflow. Orders are fulfilled on System 2 by matching physical product to order line items, managing multi-unit orders, packaging the product for shipment, and finally directing the package to an appropriate carrier. After order

fulfillment, the order status is replicated back to System 1.Figure 6 shows the order flow process.

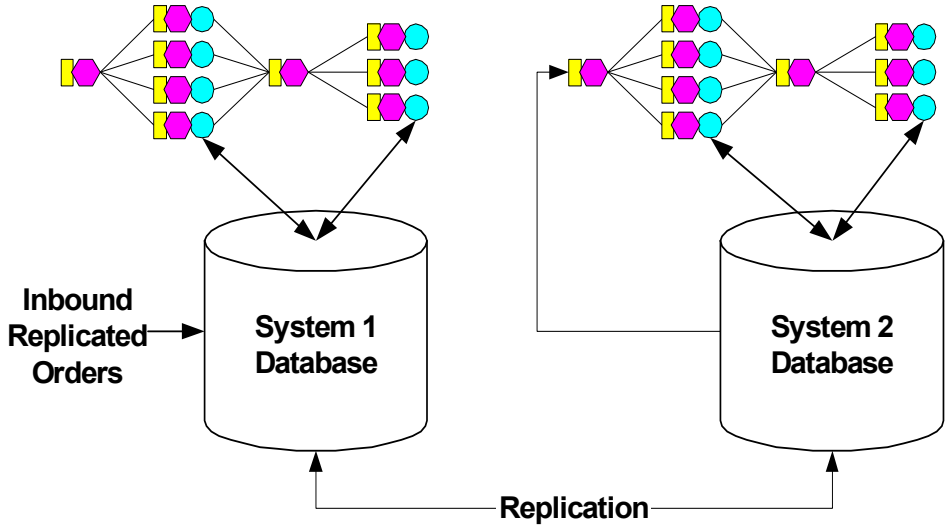

Figure 6: The order flow process

Should a catastrophe occur on one of the cluster nodes, such as CPU or power supply failure, Windows Clustering automatically restarts failed processes on the surviving node. In this way, high availability is achieved.

Under heavy system loads, such as a busy holiday season, additional queues, drainers, and COM objects can be created. As mentioned earlier, a well-implemented dispatcher might reread its control records periodically and thus dynamically allocate work to the new queues. If multiple drainers and COM objects are running and, if the servers have multiple processors, Windows 2000 distributes the load across multiple CPUs. Additionally, some of the workload can be moved to the alternate cluster node. If even more servers are needed to process orders, an additional cluster or two can be brought online and orders can be dispatched to these servers. In this way, high scalability is achieved.

If, at some point, the number of application servers saturates the database server, a distributed database solution should be explored. For example, customers and orders might be distributed across four databases. You can decrease the load against each database server by several orders of magnitude.

Summary

This paper has described the building blocks that can be used to create a scalable, highly available application using Windows DNA platform technologies.

This paper reviewed some existing scalability and availability fundamentals. It also introduced a real-world workflow architecture that can be utilized by your organization to create a scalable, high availability application running on multiple servers. Combining Microsoft technologies allows you to create the most powerful enterprise solution available.

For More Information

For the latest information on Windows DNA, please see http://www.microsoft.com/dna/about/overview/default.asp

“Building Tomorrow's Applications Today With Windows DNA 2000”: http://www.microsoft.com/dna/about/whitepapers/dna2000.asp

“How to Install SQL Server 7.0, Enterprise Edition on Microsoft Cluster Server”: http://support.microsoft.com/support/sql/content/70papers/70clstr.asp

"Introduction to Designing and Building Windows DNA Applications," Frank E. Redmond III: http://msdn.microsoft.com/library/techart/windnadesign_intro.htm

“Microsoft Message Queuing Services (MSMQ) Tips”: http://msdn.microsoft.com/library/backgrnd/html/msmqtips.htm.)

“Windows 2000 Reliability and Availability Improvement”: http://www.microsoft.com/technet/win2000/win2ksrv/technote/relwp2.asp

“Replication for SQL Server 7.0”: http://www.microsoft.com/SQL/DeployAdmin/replication.htm