


Realtime  
publishers

"Leading the Conversation"

# *The Shortcut Guide<sup>™</sup> To*



## SQL Server Infrastructure Optimization

*sponsored by*



*Don Jones*

Chapter 3: The New Cluster: Technologies for SQL Server Infrastructure Optimization .....	41
Rethinking the SQL Server Cluster .....	42
Processor and Memory Requirements .....	42
Storage Requirements .....	42
Network Requirements .....	44
Foundation Technologies for the New Cluster .....	46
Abstracting the Processor and Memory .....	46
Abstracting the Storage .....	46
Abstracting the Network .....	48
Coordination: The Key to the New Cluster .....	49
Rethinking SQL Server Management .....	51
Managing SQL Server Software .....	51
Adding Nodes to the Cluster .....	51
Adding SQL Server Instances .....	52
Rolling Hardware Generations .....	53
The New Cluster and Infrastructure Optimization .....	54
Techniques for Creating Clusters .....	54
Using Clusters to Improve Infrastructure Optimization .....	55
Benefits of the New Cluster .....	56
Consistent Software Configurations .....	56
Reduction in Operating Costs .....	56
High Availability .....	56
Reduced Complexity .....	57
Room for Growth .....	57
Conclusion: Welcome to the Optimized SQL Server Infrastructure .....	59

## Copyright Statement

© 2007 Realtimepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimepublishers.com, Inc. (the “Materials”) and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimepublishers.com, Inc or its web site sponsors. In no event shall Realtimepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtimepublishers.com and the Realtimepublishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimepublishers.com, please contact us via e-mail at [info@realtimepublishers.com](mailto:info@realtimepublishers.com).

[**Editor's Note:** This eBook was downloaded from Realtime Nexus—The Digital Library. All leading technology guides from Realtimepublishers can be found at <http://nexus.realtimepublishers.com>.]

## Chapter 3: The New Cluster: Technologies for SQL Server Infrastructure Optimization

The term *clustering*, when applied to SQL Server, is almost always used to denote a high-availability solution. In other words, SQL Server clustering is traditionally seen as a means of providing redundancy to SQL Server. Although that's certainly admirable, and in many environments desirable, it's not the end of what clustering can do. Rethinking what the word *clustering* can mean opens up the possibility of clusters that can actually help achieve an optimized SQL Server infrastructure. That's what I call the *new cluster*: Clusters that use different techniques and technologies to go beyond high availability and to instead provide the kind of agility required to create what Microsoft's Infrastructure Optimization Model would recognize as a Dynamic organization.

A number of third-party companies produce high-availability products for SQL Server, many of which use the term *clustering* to refer to groups of servers that provide backup capacity for each other. These companies include Neverfail, XLink, Sonasoft, and others. Some companies provide solutions that go beyond high-availability and seek to meet some of the other goals of infrastructure optimization, such as SQL Server consolidation; these companies include EMC, Scalent, and HP's PolyServe brand. What's interesting is that most of these companies take very different approaches to SQL Server, with the aim of achieving different business goals. For portions of this chapter, I'll use the HP PolyServe Software for Microsoft SQL Server consolidation product as an example to illustrate some of the various concepts I'll introduce.

I should note that mere consolidation is not actually a goal of what I'll discuss in this chapter; it's almost a side benefit. With a truly optimized SQL Server infrastructure, consolidation—that is, a reduced number of SQL Server computers handling the same workload—is something you get “for free.” The real goal is to create a set of techniques and technologies that help achieve a Dynamic organization, with the ability to quickly rearrange the SQL Server infrastructure, completely dynamically, to meet changing business needs.

## Rethinking the SQL Server Cluster

I want to step back for a minute and forget everything we've discussed about the often-overused word, *cluster*. Instead, I want to focus on SQL Server's actual product requirements, and discuss how those requirements can lend themselves to a new type of cluster—one that is designed to meet the goals of an optimized infrastructure rather than just high availability. I'll look first at SQL Server's three main requirements: processor and memory, storage, and network.

### Processor and Memory Requirements

For the purposes of this discussion, processor and memory are something I consider to be a single unit—perhaps this section would be better named “stuff that is attached to the server's motherboard requirements,” instead. Both are hardware resources that “live” inside the server and can't be taken outside of the server in the way that, for example, disk storage can be.

SQL Server does, of course, require processor and memory resources in order to run. Because SQL Server is a multi-instance product, the processor and memory resources of a single server are considered a pool, which is shared amongst the instances of SQL Server that are running (as well as being shared with Windows itself and any other running applications). Windows dynamically allocates these resources on the fly based on the current workload.

From a planning perspective—and ignoring Windows itself and any other applications (SQL Server computers typically don't run anything but SQL Server in terms of major applications), the SQL Server instance is the smallest unit of management when it comes to processor and memory resources. That is, you can't allocate processor time or memory capacity to a single database: Windows allocates those resources to an entire instance of SQL Server. Thus, if a single instance hosting a single database is experiencing a lot of growth in its workload, that work can't be subdivided without significant database re-design (such as federated databases, which are beyond the scope of this book).

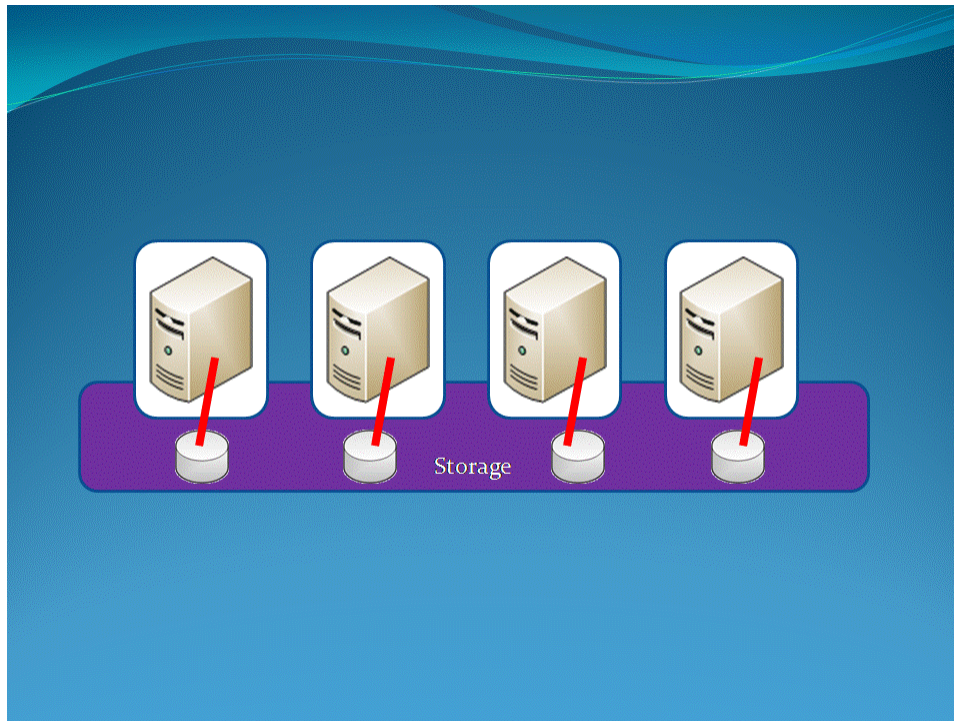
### Storage Requirements

Storage resources are much more interesting than processor and memory resources, in part because of the advent of Storage Area Networks (SANs), which permit disk storage to be moved “outside the server” (and physically away from the server), and more importantly to allow a single large pool of disk space to be broken down, or partitioned, for use by different servers. Perhaps the most typical type of SAN configuration is a single pool of disk resources, partitioned down into “chunks” that are made available for the exclusive use of a single server.

The reason that this type of configuration is “typical” is because neither Windows nor most basic SAN products have the ability to deal with resource contention. That is, if a single “chunk” of SAN disk space was made available to two servers simultaneously, things would work more or less fine provided both servers did nothing but *read* the data contained in that disk space (there are cautions you would need to take with regard to file permissions and so forth, but those are more administrative issues than technical ones). Even if one server only reads data and the other server wrote the data, things would probably work more or less fine. It's if the two servers tried to modify the shared data at the same time that things would go wrong: At best, the changes made by one server would simply be lost; at worst, the two servers would experience continual file-access errors as they attempted to modify the same data at the same time.

Interestingly, this is *not* an inherent problem when it comes to SQL Server. Consider the way SQL Server works: Each database consists of two or more files: At least a database file (with an .MDF filename extension) and at least one log file (with an .LDF filename extension). An instance is configured to open that database and make it available for client queries. However, when SQL Server actually does open a database, it *locks* that database's files. And, before SQL Server attempts to open a database, it checks to see whether the database's files are *already locked*. Thus, you could actually assign two instances, running on the same server, to both open the same database. Only the instance that actually started up first and locked the database would be able to do so; the second instance would “see” the lock from the first, and would gracefully fail to open the database.

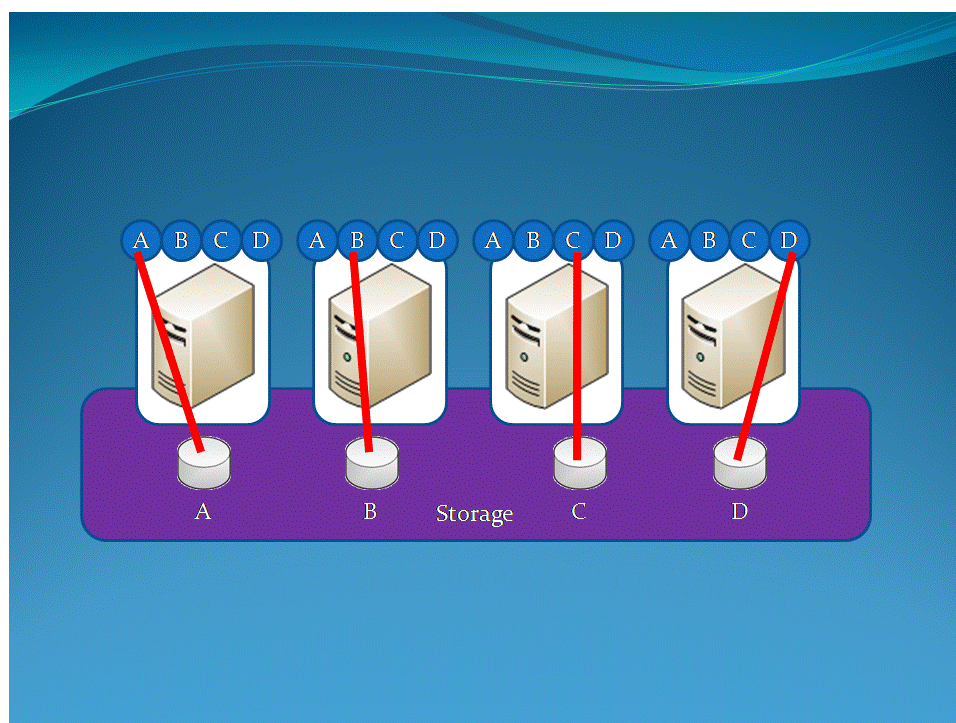
This behavior raises interesting possibilities. Figure 3.1, for example, shows four SQL Server computers connected to a single “chunk” of SAN disk space. This disk space is used *only* for storing databases; in my initial configuration, each SQL Server computer is running a single instance, which is opening only a single database. As you can see, there is no contention for disk files or other disk resources, even though all the servers can “see” the other servers' database files.



**Figure 3.1: Shared storage.**



These four servers now share a particular resource—the disk storage they can all “see”—so they can reasonably be referred to as a *cluster*. But keep in mind how SQL Server works: It’s okay to configure multiple instances of SQL Server to open the same database files because only the first one in will “win.” That makes a configuration like that shown in Figure 3.2 possible. Here, each server is running *four* instances of SQL Server. Each instance, A through D, is configured to open a single database, A through D. However, even though all four servers are “trying” to open the same four sets of files, there will be no resource contention because only the first instance to open each set of files will “win.” Obviously, relying on the “first one in, wins” method might not achieve the consistent, reliable results we’d want in terms of management, but this is a first step toward creating a new, more optimized infrastructure for SQL Server.



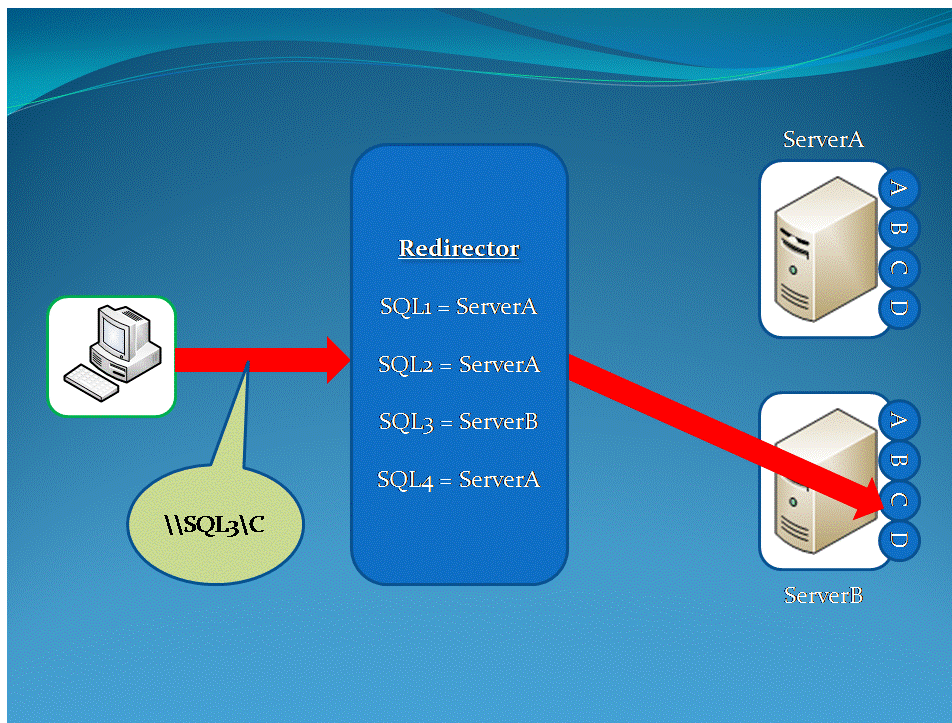
**Figure 3.2: Shared storage, multiple instances—no contention.**

### Network Requirements

When I discuss network requirements in this context, I’m not focused on the need for each SQL Server to be physically connected to a network, although that’s obviously a requirement. Instead, I’m looking at the need for client computers to connect to SQL Server via some type of address, usually an easy-to-remember name that is translated (or *resolved*) to an IP address by a DNS server.

As I discussed in the last chapter, a major hurdle to SQL Server consolidation and agility is these DNS names. If clients are connecting to ServerA and you move the database to ServerB, you’re going to have to somehow reconfigure these clients to connect to ServerB instead.

Certain tricks can be done with DNS to help abstract the connection between a name and a physical server. For example, the server name “SQL1” might not refer to a physical server name at all but might rather be a “nickname” to a server named ServerA. If you wanted clients to start connecting to ServerB instead, you’d just change the “nickname” in DNS so that “SQL1” pointed to “ServerB” instead. That trick doesn’t necessarily work with SQL Server instances, though—at least, not without some additional tricks in SQL Server itself. However, the overall *technique* is a good one, and it *can* be leveraged in SQL Server through the use of third-party software. Figure 3.3 illustrated this: A client computer attempts a connection to \\SQL3\C, a named instance of SQL Server. A combination of DNS and third-party redirector software gets the client to the correct computer—ServerB—where an instance named C is running.



**Figure 3.3: Redirector software helps abstract server names from actual servers.**

If you needed to take ServerB offline for maintenance, you’d simply notify the redirector to start sending SQL3 requests to ServerA instead. ServerA already has an instance named C installed, and provided it was configured the same as the instance C on ServerB—and could “see” the same database files—clients would be none the wiser. Provided ServerA and ServerB used a SAN disk configuration like the one I discussed in the previous section, having them each “see” the same database files would be easy, and you’d be well on your way to a more optimized SQL Server infrastructure.



## Foundation Technologies for the New Cluster

Revisiting the previous chapter, and translating some of the requirements I outlined into “new cluster” terms, we find a fairly simple set of requirements. We essentially just need to be able to move instances from server to server in response to changing workload, hardware failures, or maintenance requirements. We need to be able to do this on a “cluster” of heterogeneous (in terms of hardware) servers so that we can incorporate all the hardware currently being used to run SQL Server in the environment. For the moment, I’m not focusing on security issues—specifically, the issue of what administrators might have access to within SQL Server. That’s something I’ll address later. For now, let’s focus on the basic infrastructure technologies necessary to meet the goals I’ve stated.

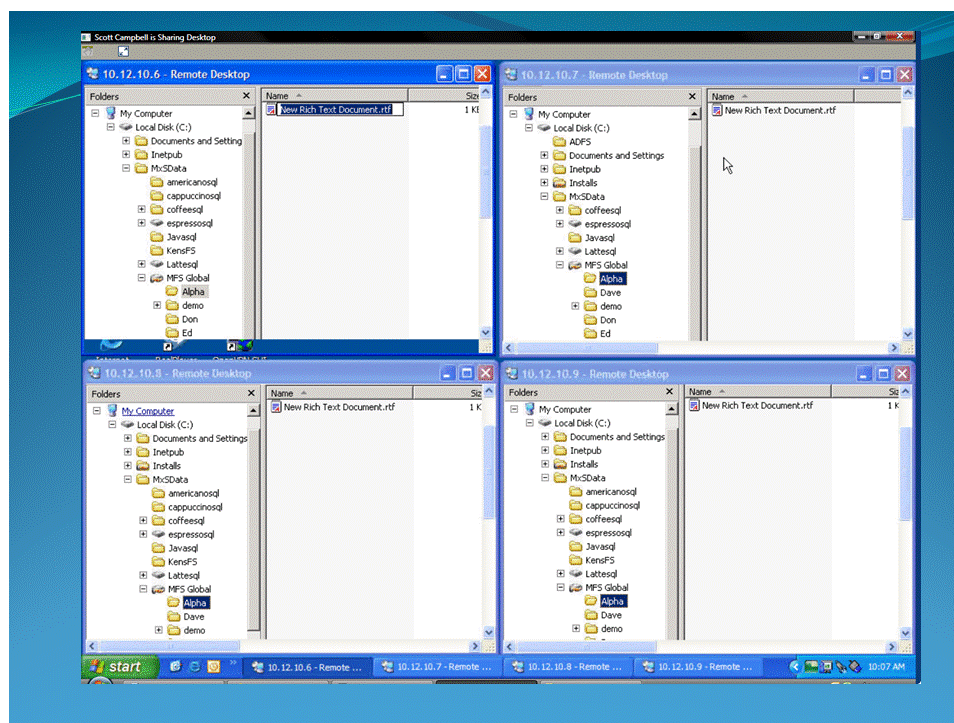
### ***Abstracting the Processor and Memory***

In one sense, it’s impossible to “abstract” processor and memory resources. That is, there’s no way to take the processor and memory resources of several servers and have them appear as one giant “pool” of resources. However, SQL Server instances don’t “care” what processor, or how many processors, they run on, so from that perspective, abstraction isn’t necessary. That is, a given SQL Server instance can be installed on multiple servers and can be started or stopped on those servers as necessary. So long as the instance can “see” the database files it’s configured to use, and so long as clients can connect to the instance, the instance doesn’t “care” what server it’s running on.

This is similar to the way the Windows Cluster Service operates in an “active-active” configuration. Each cluster node runs one or more SQL Server instances, with the capability to move instances to a different node on demand. The main difference between the “new cluster” and Windows Cluster Service is in how the cluster deals with disk storage.


### ***Abstracting the Storage***

SANs make it fairly easy to abstract the network. As I explained earlier, the right SAN solution will allow the same “chunk” of disk space to be assigned to multiple servers. All the servers will be able to “see” all the files in that disk space, and provided there’s no need for resource access contention management—which there isn’t, with SQL Server—there won’t be a problem. Figure 3.4 illustrates this: You’re looking at four Remote Desktop sessions, each connected to a different server. All four servers have an “MFS Global” volume, which exists on a SAN. The disk space for that volume is the same physical space for all four servers, meaning that when I create a “New Rich Text Document.rtf” file on one server, it immediately is “seen” by the other three.



**Figure 3.4: Abstracting storage through a SAN.**

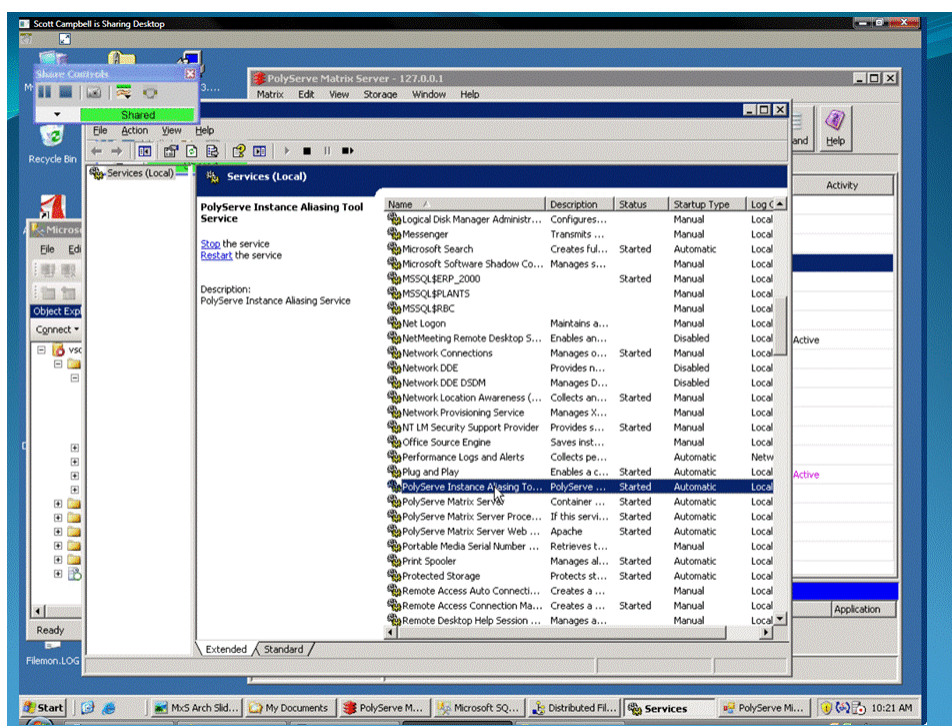
With a design like this, it doesn't matter how many servers are in my "cluster." Provided all of my SQL Server database files exist on that SAN disk space, any given server in the "cluster" can "see" any given database, meaning any SQL Server instance could be started on any given server, and it would run fine.

 Note that additional disk space is needed for the Windows OS and the SQL Server application files—this space is often in fault-tolerant storage inside the server, although it could also be on other forms of storage. That "system" disk space needs to be private for each server because those files can't be easily shared due to resource access contention.

This design is very different from the approach used by the Windows Cluster Service. Windows clustering, when used with SQL Server, requires that a given cluster node exclusively "own" disk storage that can be "seen" by every node in the cluster. In other words, if Node A is running SQL Server Instance A, then Disk Storage A must be exclusively available for Node A's use, and it's where the instance will look for its database files. If the instance is moved to Node B, then Node A must "relinquish" the disk space and allow Node B to have exclusive use of it. Any other instances that were running on Node A and storing their files in the same disk space must also move to Node B because "their" disk storage is moving. Windows Cluster Service's design thus limits and complicates how flexible the cluster can be: Either each instance must have dedicated disk storage—creating a complex system of resources and dependencies that must be managed—or you must be content moving instances in groups according to their disk storage. The "new cluster" is more flexible because storage is not a logistical issue because it's been abstracted away from a particular server or SQL Server instance.

## Abstracting the Network

As I mentioned earlier, special software will be needed so that clients can address specific SQL Server instances using names that aren't tied to a specific physical server. As I explained, DNS tricks alone aren't enough due to the way SQL Server works; instead, specific third-party software is needed to create nicknames, or *aliases*, for SQL Server instances. As Figure 3.5 shows, HP PolyServe Software provides an Instance Aliasing service that handles this requirement. It allows an administrator to define a series of abstract “server names.” In the organization's DNS, these names are “pointed” to IP addresses that have been assigned to an HP PolyServe Software “cluster” (that is, a collection of computers all running this aliasing service, amongst other things). The aliasing service takes care of translating those incoming “server names” into actual physical server names and SQL Server instance names so that the client is connected to the proper SQL Server instance, no matter where in the cluster that instance happens to be running.



**Figure 3.5: Aliasing abstract names to server/instance names requires a third-party service.**

This is actually similar to the way that Windows Cluster Service works, with the Service itself providing the aliasing functionality.

### Coordination: The Key to the New Cluster

Let me wrap up what the “New Cluster” looks like so far: It’s a collection of physical servers, each with several instances of SQL Server installed. In fact, at this point, let’s say that every server has a copy of every SQL Server instance that the cluster is capable of running, meaning that any given instance can run on any given server. That’s about the best we can do in terms of abstracting processor and memory resources: Ensuring that any given instance can run on any given processor/memory set within the cluster.

The servers are all connected to a SAN, and they have at least one volume that they share. This allows every server to “see” every database’s files, effectively abstracting the disk storage so that it’s not server-dependent.

The servers all run an aliasing service that allows them to translate abstract host names into physical server/instance names. This allows clients to address a specific SQL Server instance, without being aware of the physical server that is running that instance at a given time.

Our cluster still has a few things that need to be done in order for it to work effectively: First and foremost, some form of coordination needs to exist so that only one server attempts to start any given SQL Server instance at a time. That way, we can *predict* which SQL Server computer will be running any given instance, and we can update our aliasing service accordingly, so that it “knows” where to send incoming requests for that particular instance. The HP PolyServe Software for Microsoft SQL Server product (which incorporates the aliasing service I pointed out earlier) provides this coordination. Figure 3.6 shows the main HP PolyServe Software administrative interface.

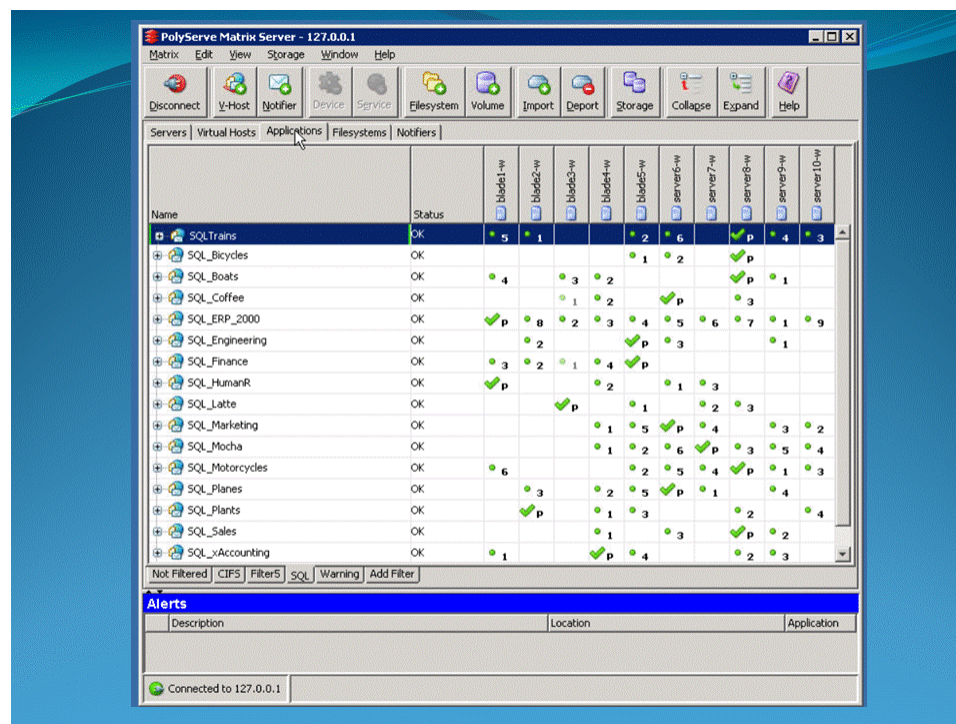


Figure 3.6: HP PolyServe Software for Microsoft SQL Server.

As you can see, this particular product doesn't require that every server be able to run every SQL Server instance within the cluster. Servers are shown across the top of the grid, and SQL Server instances down the left side. A green dot within the grid means that server is capable of running that particular instance of SQL Server; the number indicates the server's "preference" for that instance. A preference of "1" means that the server is the top choice for running that instance, for example. A green checkmark indicates the server that is actually running a given instance of SQL Server. This product provides the coordination necessary for the cluster to function, ensuring that each SQL Server instance is only started on a single physical server at any given time. It also provides other administrative and management functionality, which I'll get to in the following section. At this point, though, we have a cluster fully capable of meeting the requirements for a highly optimized SQL Server infrastructure:

- Any instance of SQL Server can be started on any given computer. Thus, we can take a server offline for maintenance without significant downtime because its instances can be transferred to another computer.
- Free resources on any server can potentially be used for any instance. Although all of our free resources aren't exactly "pooled" (meaning that two servers with 50% capacity don't equal one server with 100% capacity because the two servers couldn't "split" a workload requiring 75%), our free resources can be much more flexibly used by whatever workload will fit.
- A hardware failure is no more a problem than hardware maintenance: We simply transfer SQL Server instances to other servers, taking the failed server out of the cluster until it's repaired.
- We don't need homogenous hardware within the cluster. Because SQL Server instances don't care what they're running on (provided the server meets the minimum SQL Server requirements, of course), we're free to use any mix of servers that meets our requirements. That means we can create our "new cluster" using existing SQL Server resources.

We still haven't solved all the management problems—patch management, agile deployment of new servers, and so forth—that we need to be a fully Dynamic organization. I'll tackle those topics next.



## Rethinking SQL Server Management

What I've introduced so far of the “new cluster” points toward a *more* Dynamic organization, but not a *completely* Dynamic one. There are still problems that I haven't addressed, primarily related to cluster and server management. A Dynamic organization would manage the entire cluster as a unit, looking at overall cluster-free resources, adding nodes to the cluster almost on demand, and so forth. So the next major element of the “new cluster” is to rethink SQL Server management.

We can no longer be satisfied with managing individual servers, reconfiguring individual servers, and maintaining individual servers. Doing so simply leaves us stuck with the same problems as managing non-clustered SQL Servers: Every new server we add to the mix creates additional administrative overhead. In order to become a Dynamic organization, we need to be able to maintain more or less the same amount of administrative overhead for the entire cluster, no matter how many servers it encompasses.

### Managing SQL Server Software

The “new cluster” allows an entire cluster to be managed as easily as a single server. Obviously, database-specific configurations are stored within the SQL Server database itself, and so any SQL Server instance that opens that database will “see” any configuration changes. Server-level configuration for individual instances can be managed through the HP PolyServe Software console. These configuration changes can then be “pushed” out to the individual servers automatically by the HP PolyServe Software. Configuration changes can include updates and hotfixes along with instance-level configuration such as port number and other settings.

### Adding Nodes to the Cluster

A major requirement of the “new cluster” is the ability to quickly add new nodes in order to support additional workload. New nodes can either be intended to support growth in the cluster's workload or can be intended to replace older, outdated hardware. HP PolyServe Software helps provide a Dynamic level of functionality by providing a single console that can “push” SQL Server's software installation out to new cluster nodes—including multiple nodes at once. Figure 3.7 illustrates this concept, with nodes being selected to receive the SQL Server software. Service packs can be deployed in the same fashion, helping to simplify long-term maintenance of the cluster. Overall, this functionality allows new nodes to be added to the cluster much more quickly and helps to remove the “individual server” management technique in favor of a “cluster-wide” management technique.



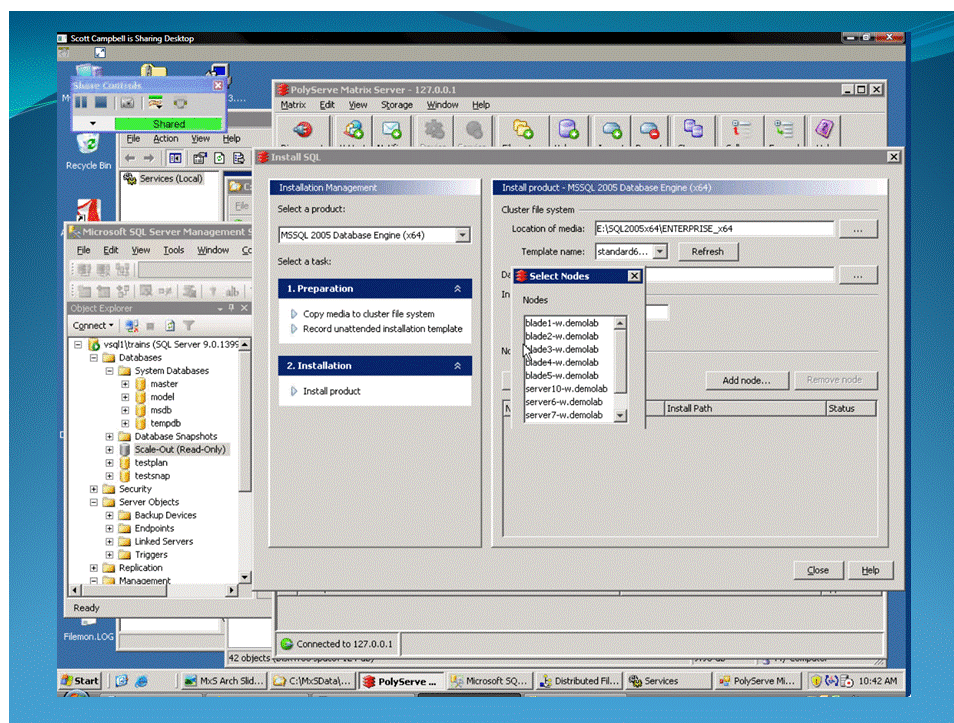
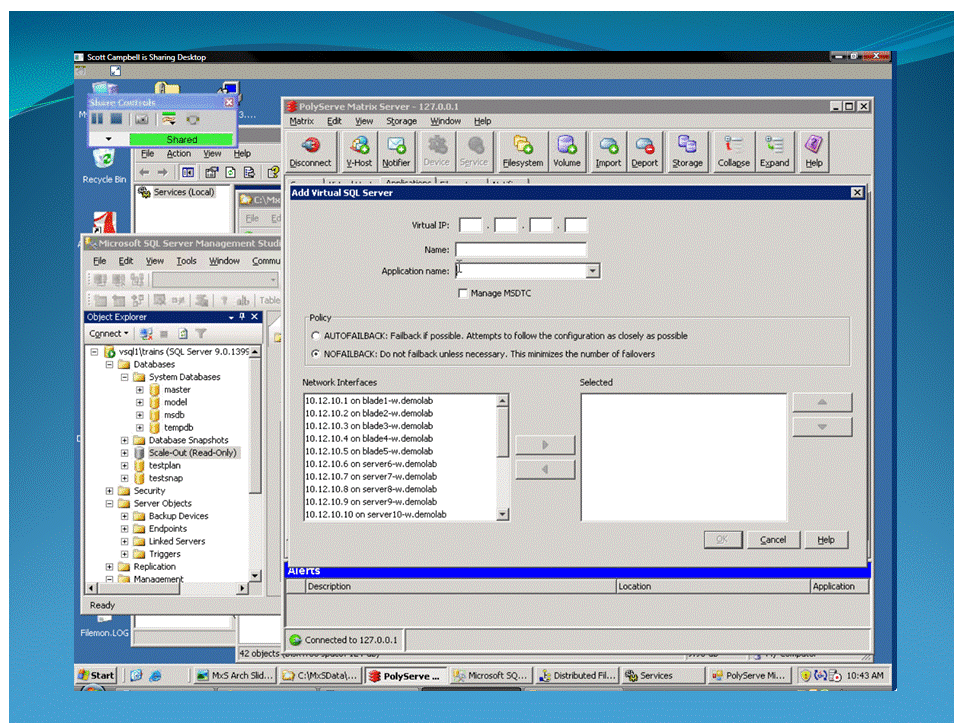


Figure 3.7: Deploying SQL Server to new cluster nodes.

### Adding SQL Server Instances

Adding new SQL Server instances to the cluster should also be a one-step operation, with the new instance configured on whatever cluster nodes are desired automatically. Figure 3.8 shows this task in HP PolyServe Software: Each new instance—called a *virtual SQL Server* by this product—gets a dedicated IP address, a name, and an application name. As many cluster nodes as desired can be designated to potentially host this instance, and a failback policy (which determines whether the instance will try to “fail back” to its preferred cluster node after “failing over” to another for some reason) can be configured to support high availability.



**Figure 3.8: Configuring a new SQL Server instance in the cluster.**

## ***Rolling Hardware Generations***

Another major benefit of the “new cluster” is the ability to use dissimilar hardware for cluster nodes. This allows all existing SQL Server hardware to potentially become a resource within the cluster. And, because the cluster allows nodes to be almost transparently moved from node to node, it’s relatively easy to remove old, unsupported hardware from the cluster. When an older server is beyond its operational life, you can “roll” the cluster’s hardware generation by adding a new node to the cluster, moving SQL Server instances to it (thus “emptying” the old server), and simply removing the old server from the cluster. Minimal downtime is required—usually on the order of a few seconds for each SQL Server instance you move—making it easy for nearly any organization to find an appropriate maintenance window.

## The New Cluster and Infrastructure Optimization

The “new cluster” provides all the tools you need to create a fully optimized SQL Server infrastructure, one that Microsoft’s Infrastructure Optimization Model would recognize as Dynamic. Of course, simply having the technologies in place doesn’t give you that optimization: You have to use them correctly.

### Techniques for Creating Clusters

It’s unlikely that large organizations will lump every SQL Server they own into a single cluster. After all, different SQL Server computers serve distinctly different functions, and you still have to accommodate factors such as large, geographically distant offices that require their own physical servers. Exactly how you choose to build your clusters will depend on your exact business requirements, and it’s a question that bears careful consideration and planning. Figure 3.9 shows one simple example of how an organization’s SQL Server computers might be “divvied up” into three different clusters.



**Figure 3.9:** One way to divide servers into clusters.

Here, I’ve created the largest cluster for my organization’s line-of-business applications. This might include my SAP Financials database, my PeopleSoft databases for personnel management, as well as several in-house databases used for various business-specific applications. This cluster will use my newest and best server hardware, and it might have a fairly large overall amount of free capacity so that a significant number of servers could fail entirely without taking a single database offline for more than a few seconds (of course, if the maximum number of servers did fail, I might be satisfied with less-than-optimal application performance, so I don’t need the cluster to have 200% of my working capacity).

A second, smaller cluster is used to run smaller department- and project-specific databases. This cluster might use older hardware and might have much less overall free capacity depending on how critical these smaller applications are. A third, very small cluster might be used for development and testing purposes.

Workloads couldn't be easily moved *between* these clusters; doing so would be far from impossible, but it would require some manual effort, file copying, and so forth. Workload could be repositioned within the cluster, though, based on hardware failures, maintenance, and other needs. If a department-specific application one day became mission-critical, and I decided to move it to my large cluster, doing so wouldn't be a massive undertaking: I'd use the HP PolyServe Software migration tools to automatically create a new SQL Server instance on the large cluster while maintaining configuration and security settings and safely copy the database files between the two clusters' shared storage pools, providing a working mirror copy of the first database without ever taking the database out of service. I'd decide which nodes of the new cluster would be potential hosts for the new SQL Server instance, and once testing is complete, I flip the switch to migrate over any incremental updates to the primary database and transition users to the new server. This is the type of dynamic flexibility that the Infrastructure Optimization Model calls for in the Dynamic level.

### ***Using Clusters to Improve Infrastructure Optimization***

Let's look at the "new cluster" in terms of Microsoft's Infrastructure Optimization Model—specifically, the Dynamic level (characteristics are quoted from <http://www.microsoft.com/technet/infrastructure/datasheet.mspx>):

- "Processes are fully automated." With regard to SQL Server, this is true. Failover occurs automatically within the "new cluster," and other processes—deploying software updates, deploying new SQL Server instances, manually moving SQL Server instances, and so forth—are all automated.
- "The use of self-provisioning software..." is definitely in place. When the time comes to create a new SQL Server instance or to add a new node to the cluster, you just point the clustering software at the task and it handles it.
- "...with service level agreements...established." This is also possible with the "new cluster." Although you could certainly establish service level agreements (SLAs) without this clustering technology, actually meeting those SLAs requires a much larger hardware and administrative investment. With the "new cluster," you can meet your SLAs with minimal redundancy and with minimal additional administrative overhead.
- "Costs are fully controlled." This is definitely possible because the cluster provides you with the desired level of high availability without excessive redundancy, and without imposing significantly higher administrative overhead. In other words, you can achieve business goals for availability and maintenance without vastly increased costs from hardware or administrative overhead.

The "new cluster" gives you the capabilities to meet all these criteria, helping to make SQL Server a more strategic asset within your organization.

## Benefits of the New Cluster

The “new cluster’s” use of shared disk resources, alias-based connectivity, single point of administration, and flexible reallocation of workloads provides a number of business benefits.

### ***Consistent Software Configurations***

SQL Server software stays consistently configured. From a security and operational standpoint, that is a huge benefit; you can be assured that no matter which server is presently hosting a given SQL Server instance, it’ll behave exactly the same as any other server that might host that same instance. For organizations subject to industry or regulatory compliance requirements, this consistency helps make auditing easier and more problem-free. Consistency also helps reduce debugging time for in-house developers because they’re assured of the same server behavior at all times and don’t need to be concerned about different nodes behaving differently. Consistency also applies to SQL Server updates and hotfixes, ensuring that every instance of SQL Server is at the same patch level at all times, no matter which cluster node the instance happens to be running on at the time.

### ***Reduction in Operating Costs***

Because maintenance—including instance reconfiguration and patch deployment—doesn’t need to be performed on a per-server basis, administrative costs are significantly lowered. In fact, using a product such as HP PolyServe Software can help reduce administrative costs immediately, even if you don’t take advantage of the product’s other features and capabilities.

Capital costs can also be reduced because individual servers don’t necessarily have to be configured with vast amounts of inflexible extra capacity. Instead, free capacity can be provided to the cluster as a whole, giving you a place to move SQL Server instances that are outgrowing their current server, and giving you the ability to bring in new server resources—and remove older hardware—on a gradual, as-needed basis.

### ***High Availability***

Availability is built-in to the cluster, providing full communication between cluster nodes and immediately relocating SQL Server instances from failed servers to running ones. The instances from a failed server can be relocated to many different “backup” servers, if necessary, allowing you to customize the impact on application performance by re-balancing the workload automatically through instance preference assignments. Instances can be configured to “fail back” to their preferred server automatically if the server becomes available again after a failure.

SQL Server’s own fault tolerance ensures that, when an instance fails over to another cluster node, data loss is minimal. The new node has access to the same SQL Server data and log files, allowing it to immediately begin transaction recovery as it begins to accept new client connections.

Although the disk storage shared by the cluster’s nodes can be seen as a single point of failure, in reality, most SANs provide many options for fault tolerance, including RAID arrays, hot-swappable drives, and more. You can also schedule frequent SQL Server backups to a second “shared” disk storage area, allowing SQL Server instances to (for example) create very frequent log backups to guard against physical damage to the database files.



### Reduced Complexity

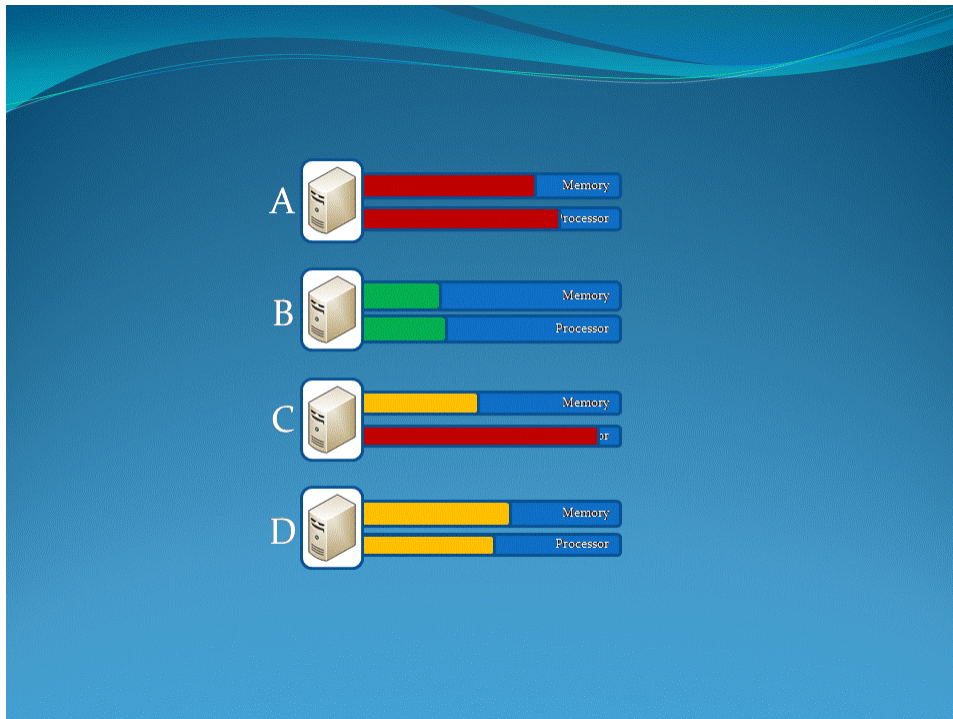
Although managing a cluster full of SQL Server computers may *sound* more complex, it actually offers less complexity than managing a similar number of individual, non-clustered servers.

Because much of the cluster's management is done from a single location, managing many aspects of the cluster is no different from managing a *single* SQL Server computer—no matter how many nodes exist in the cluster.

And the “new cluster” offers less complex management than Windows Cluster Service. SQL Server instances don't have complex disk storage dependencies that have to be managed. Instead, each SQL Server instance is an entity unto itself, with only a network alias name that has to be “moved” with the instance—and the clustering software takes care of that automatically. Disk storage resources are shared across the cluster, ensuring that every instance can “see” its disk storage, no matter which cluster node the instance happens to be running on.

### Room for Growth

One of the main benefits of the “new cluster” is the retention of “room for growth.” In previous chapters, I described how organizations see to maintain this room on individual servers by always over-provisioning servers so that no individual server is running at 100% capacity. That creates a lot of “extra capacity” within the environment, as Figure 3.10 shows. However, that extra capacity isn't very flexible because it can *only* be used to support whatever applications are on that individual server.



**Figure 3.10: Provisioning inflexible extra capacity on a per-server basis.**



This figure illustrates four servers, each with varying levels of memory and processor utilization (there are, of course, other resources that play into overall server utilization, but I'm choosing these two to keep the illustration simple). As you can see, Server B has plenty of extra capacity, but it's useless to Server A, which is beginning to reach its maximum. Overall, this organization has something in excess of 50% free capacity—or, in other words, wasted resources—although this capacity can't be flexibly reallocated to help support a single server.

In Figure 3.11, I've made those four servers nodes in a cluster, using the new techniques I've discussed in this chapter. This enables me to relocate the workload from Server D to Server B. Now, Server D is completely unused—essentially, I've centralized most of the four servers' free capacity onto a single server by more fully utilizing Server B.



**Figure 3.11: Aggregating free capacity in a new cluster.**

Now, Server D's free capacity can support any of the other three servers. For example, if Server A's workload begins to exceed that server's hardware capacity, a beefier Server D could be used by simply relocating Server A's SQL Server instances to Server D. If Server A were running more than one SQL Server instance, then Server A's workload could be subdivided, and a portion relocated to Server D to provide room for growth on Server A. Or, if the workloads managed by Servers A, B, and C were relatively static, Server D could be removed entirely and put to some other use. Regardless of what you choose to do and how you choose to architect your cluster, the point is that you now have a flexibility you never had before, with the ability to quickly and easily move workloads across servers to create the exact balance you want.

## **Conclusion: Welcome to the Optimized SQL Server Infrastructure**

You're no longer worried about a server hardware failure, because you've got ample free capacity in your cluster and you know you can quickly redistribute SQL Server instances as necessary if a failure occurs. Maintenance is a no-downtime affair, now, because SQL Server instances can also be relocated on demand to servers with free capacity. New nodes can be added to the cluster with a few button clicks, and new SQL Server instances can be created and provisioned with a few more. You're no longer worried about processor, memory, or disk storage constraints because those resources can be easily expanded within the cluster at any time—without significant increases to administrative overhead. Your SQL Server infrastructure can respond almost instantly to changing business needs without all-night maintenance marathons and without having double the amount of SQL Server hardware you really need. Welcome to the Optimized SQL Server Infrastructure.

## **Download Additional eBooks from Realtime Nexus!**

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