


Realtime
publishers

"Leading the Conversation"

The Shortcut Guide[™] To



SQL Server Infrastructure Optimization

sponsored by



i n v e n t

Don Jones

Chapter 2: Optimizing Your SQL Server Infrastructure: Good Ideas, Bad Ideas	20
The Nature of RDBMS and SQL Server	20
SQL Server Instances.....	20
Processor and Memory Requirements	22
Storage Requirements	25
Client Connectivity Requirements.....	26
Disaster Recovery Requirements.....	27
Basic Server Consolidation.....	29
Technology Overview.....	29
The SQL Server Connection.....	29
The Impact on Sprawl.....	30
The Impact on Flexibility.....	30
Windows Server Clusters.....	31
Technology Overview.....	31
The SQL Server Connection.....	32
The Impact on Sprawl.....	32
The Impact on Flexibility.....	33
Hardware Virtualization.....	34
Technology Overview.....	34
The SQL Server Connection.....	35
The Impact on Sprawl.....	35
The Impact on Flexibility.....	36
Pooled-Resource Clustering.....	37
Technology Overview.....	37
The SQL Server Connection.....	39
The Impact on Sprawl.....	39
The Impact on Flexibility.....	40
Infrastructure Optimization: The Positive Impact on SQL Server Sprawl.....	40
Coming Up Next.....	40

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.

[**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 2: Optimizing Your SQL Server Infrastructure: Good Ideas, Bad Ideas

In the previous chapter, I discussed the impact of sprawl in a SQL Server infrastructure. Basically, my conclusion was that traditional SQL Server architectural techniques, along with a number of business factors (such as political factors) seem to make SQL Server sprawl inevitable. The result is a highly *unoptimized* infrastructure, where copious amounts of unused resources exist but which cannot be easily deployed to help fill any need. That is, if you take a look at your infrastructure as a whole, you probably have lots of unused storage, processor capacity, and so forth, but you're not necessarily free to “move” unused resources from one area to cover a shortage in another. Your resources are inflexible, tied to individual servers, which are often handling a very specific workload.

This chapter looks a bit harder at *why* things turn out this way, from a SQL Server technical viewpoint, and explores some of the ways the industry has tried to solve the problem. Remember that the ultimate goal is to create an infrastructure that's *dynamic* according to the Microsoft Infrastructure Optimization Model (IOM)—capable of changing smoothly and quickly to meet business needs, and in fact *designed* to accommodate rapid changes.

The Nature of RDBMS and SQL Server

SQL Server itself had traditionally defied attempts to be incorporated into a dynamic infrastructure. The very nature of SQL Server—the way it works—doesn't seem, at first glance, to accommodate rapid change. To be fair, the challenges you get with SQL Server are really issues with *any* relational database management system (RDBMS), whether IBM DB2, Oracle, or any other brand. An RDBMS is a resource-hungry, intensive application, and it can be very difficult to architect that type of application so that it can accommodate rapid change. Let's look at a few specific aspects of SQL Server (again, most of which apply to any RDBMS) that would seem, on the face of it, to make a dynamic infrastructure more difficult.

SQL Server Instances

Since SQL Server 2000, SQL Server has been built as a multi-instance product. In other words, you can basically have multiple copies of SQL Server running on the same computer, at the same time. Each copy is called an *instance*, and each instance has an independent server configuration and one or more user databases, as illustrated in Figure 2.1.

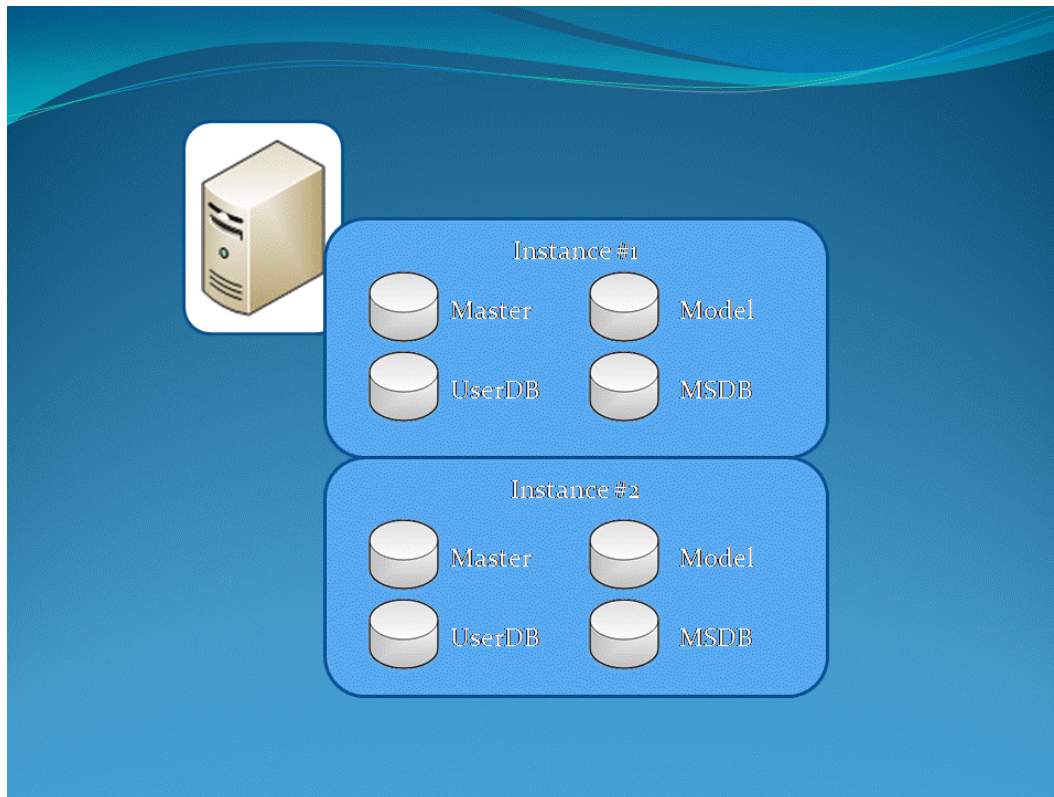


Figure 2.1: Multiple instances of SQL Server running on a single computer.

Clients connect to each instance using a combination of the physical server name and the instance's name: `\\ServerA\Instance1`, for example. Apart from this unique naming convention, client computers aren't really aware that they're connecting to an instance; as far as they're concerned, each instance could be running on a completely different physical machine.

Instances are actually a SQL Server element in *favor* of a dynamic infrastructure, although few companies really use them that way. One of the primary reasons that instances were introduced to the product was to help improve stability. If one instance needs to be taken down for maintenance, for example, another instance could continue to run, provided the physical server has sufficient free hardware resources. Another reason for instances was configuration: SQL Server has certain configuration parameters that are server-wide (well, *instance-wide*); instances permit multiple databases to run on a single physical machine, using different "server-wide" configuration settings. For example, one instance might be configured to use Windows Authentication, while another might allow mixed Windows and SQL Server authentication.

Instances also have a real security benefit because they represent a security boundary. Being able to log on to Instance A, for example, doesn't imply anything about your ability to log on to Instance B.

The ability to have instances in SQL Server actually enables almost every technique I'll examine for creating a dynamic infrastructure. That's because instances, for the first time, created an abstraction between the physical server and the "SQL Server" that client computers see and connect to. As far as clients are concerned, "SQL Server" is an *instance*, not a physical computer. If an instance was moved from one computer to another, all the client might have to do is change the name used to connect to that instance (and playing tricks with DNS could even eliminate that need). In other words, SQL Server clients don't care what physical hardware they're talking to, they care what *instance* of SQL Server they're talking to.

So, instances: A mark in SQL Server's favor for dynamic infrastructure.

Processor and Memory Requirements

SQL Server is definitely an application that is able to fully utilize a lot of processor power. Record-setting SQL Server benchmarks are often accomplished on amazingly well-equipped servers with eight or more processors running in parallel.

SQL Server is also known as an application that likes its memory. The more memory, the better, I always say, and SQL Server makes excellent use of however much memory you can throw at it. One of the major advents of the past few years has been the introduction of 64-bit processors that provide a much larger address space for memory than the traditional 4GB limit. In the past, I never built a SQL Server that didn't have the full 4GB of memory, and now with 64-bit processors, I like to throw in a few extra gigabytes of memory whenever possible.

Take some processors or memory away from SQL Server, and its performance can suffer. More importantly, either processor or memory capacity is often a "ceiling" that limits a SQL Server instance's ability to handle additional workload (the other main "ceiling" is often disk throughput, and that's the one you often hit first). Once a computer's processor and memory capacity is maxed out, and SQL Server is fully utilizing those resources, SQL Server simply can't grow any further. This single fact is the primary cause of SQL Server sprawl in many organizations (see Figure 2.2).

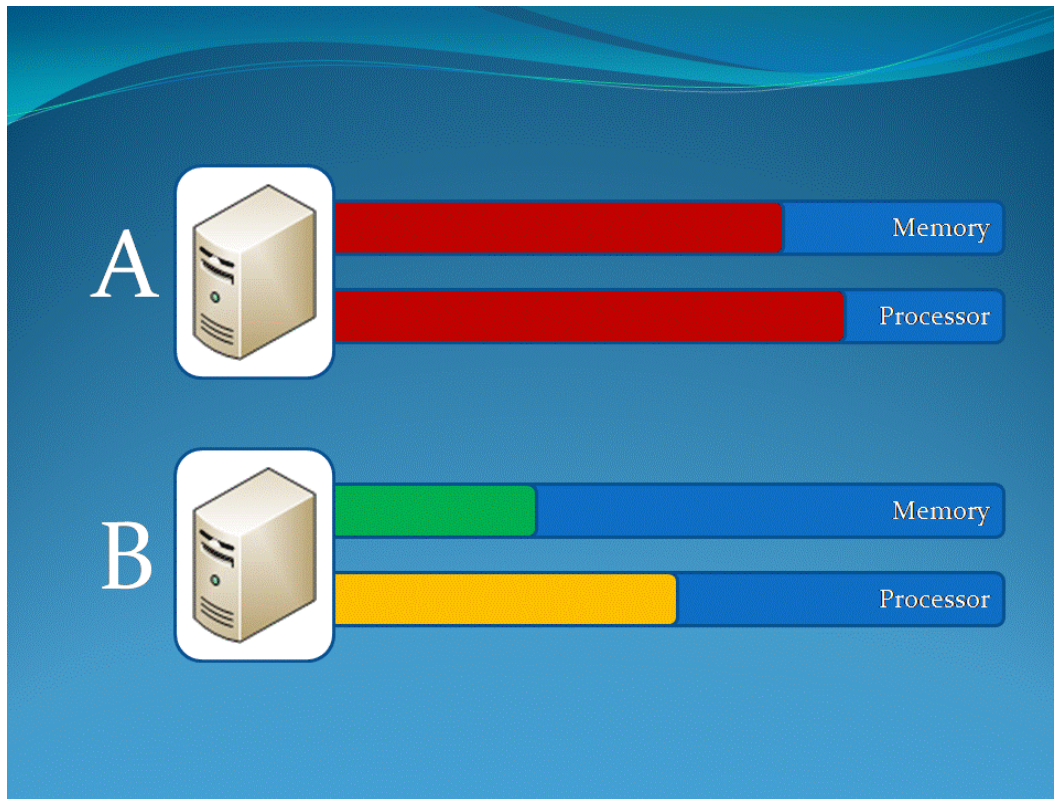


Figure 2.2: Processor and memory capacity on two SQL Server computers.

Suppose you were responsible for these two servers, A and B. The colored bars represent current resource utilization for both memory and processor. Your organization needs to deploy a new database related to a specific project, and you know that, at least initially, this new database won't use more than 5% of a SQL Server's memory or processor capacity. To which server would you deploy the database?

Most administrators would choose either "B," or a third answer: Get a new server. Because A is already nearing its limit, most would fear that either the new database, or whatever databases A is currently hosting, would eventually grow and exceed the server's capacity. Because moving a database to a new server can be a time-consuming task, depending upon how clients are connecting to that server, most administrators prefer to stick a database somewhere where they can leave it more or less permanently. Moving a database also entails downtime, which often isn't desirable from a business perspective, providing another reason to avoid hosting a database on a server that might soon run out of capacity.

This attitude on the part of administrators—largely driven by the way SQL Server itself works, and the relative difficulty of relocating databases to other servers—results in lots of servers with spare capacity sitting around. It's also what drives the installation of new physical servers, even when the infrastructure technically has enough free capacity to handle whatever new databases are needed: The anticipation of growth.

Let's say, however, that you decided to put the new database on Server A. Eventually, that short-term project database became a long-term mission-critical application (of course), and its usage grew—eventually exceeding the processor capacity of the server, as shown in Figure 2.3.

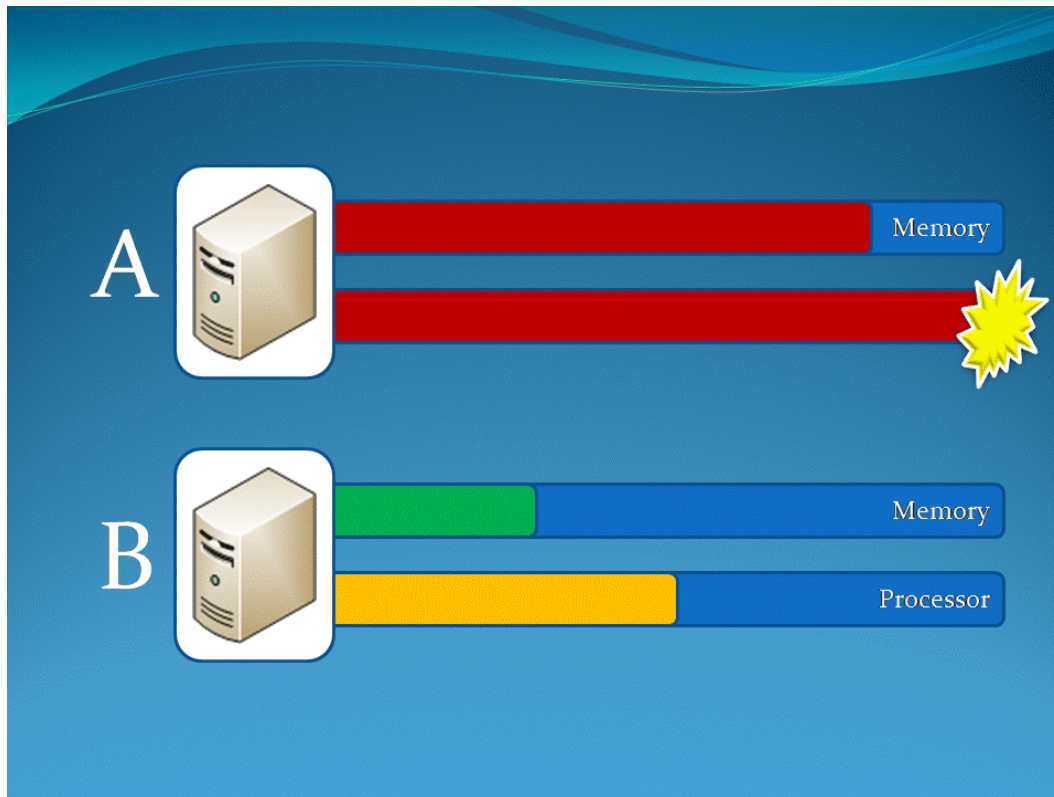


Figure 2.3: Server's processor capacity is exceeded.

Here's the ultimate problem with a non-optimized infrastructure: Even though Server B has plenty of excess capacity, it doesn't do Server A any good. Certainly, if Server A's workload is the result of more than one database, as we've suggested, then perhaps one of those could be migrated to Server B. If both databases were held by a single SQL Server instance, that migration can be fairly difficult. Client applications will have to be reconfigured or modified to connect to the different server, at the very least, and the database will experience downtime during the migration period. If the databases were hosted by separate instances, the move might be a bit easier, but there would still be a production impact of some kind.

Instead, many organizations will take an easier route. Move *all* of the workload from Server A to Server C, a brand-new, bigger, and better server. That attitude makes server manufacturers very happy, but it's not using the infrastructure very efficiently. Perhaps Server A could be redeployed to handle other databases, meaning the infrastructure finds itself with *more* inflexible excess capacity.

Storage Requirements

SQL Server also, of course, requires disk space for its databases. Disk space on large SQL Server computers these days is often provided by a Storage Area Network (SAN), which is often pretty easy to expand, giving SQL Server computers extra disk capacity when they need it. That's not always the case, of course. Many departmental or project servers might have local SCSI storage, which can be more difficult to expand on demand. In those cases, storage becomes another constraining resource much like memory or processor, which I've already discussed, and imposes many of the same limitations on the infrastructure.

But even more easily expanded SANs can lead toward a less-dynamic infrastructure, simply because of the way SQL Server must use its storage. Keep in mind that a given instance of SQL Server, no matter how many databases it is hosting, needs a single set of storage resources. This is definitely easier to explain with an illustration, so take a look at Figure 2.4.

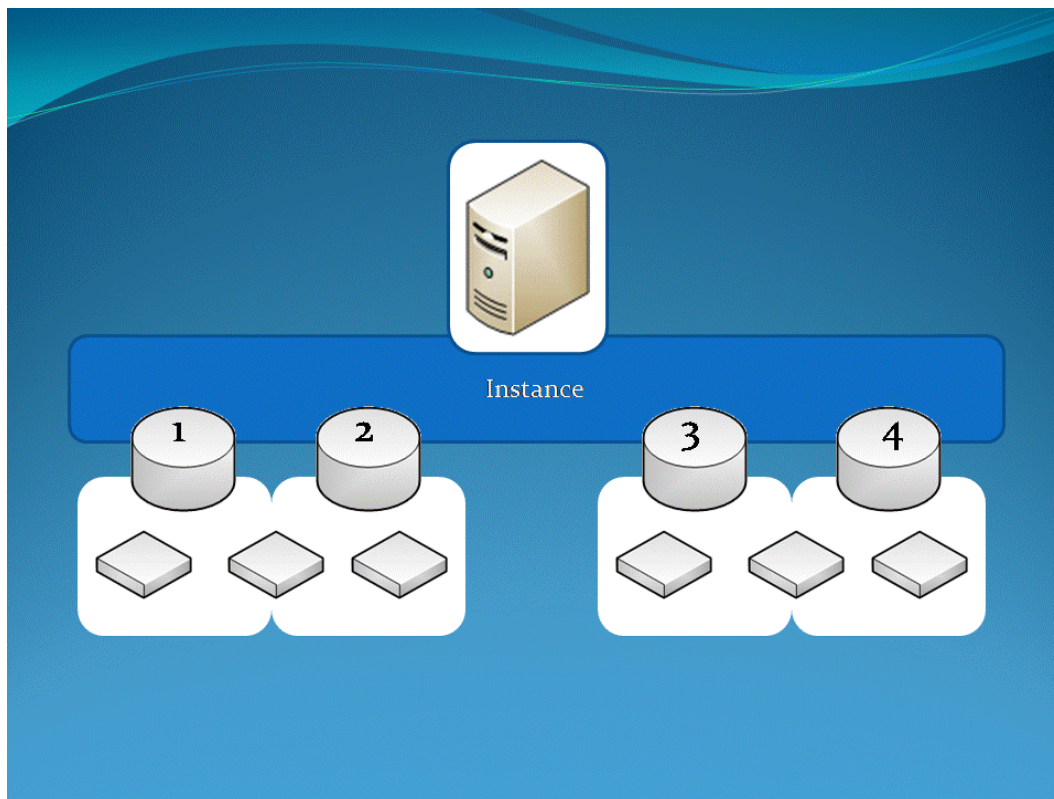


Figure 2.4: Storage resources used by a SQL Server instance.

In this scenario, a server hosts a single instance of SQL Server, which in turn hosts four databases. The server has six storage resources connected to it—logical disks, essentially, either connected from a SAN or directly from SCSI storage. Each database has its own dedicated logical disk (probably for data files), and each pair of databases shares a logical disk (perhaps for log files). In reality, of course, SQL Server storage configurations can be much more complex, but this serves to illustrate the point, which is this: *All six storage resources are tied to the SQL Server instance.* If this server starts to reach its capacity, you can't simply move the instance to a new server without also moving (or re-provisioning) all six storage elements—or reconfiguring the databases to use a different storage configuration. If one of the storage elements has a lot of excess capacity, it can't be used by SQL Server instances running on other servers, because the storage is tied to *this* instance.


This is a very important distinction: Storage is most commonly tied to the *instance* not to the *database*. Yes, the storage is being *used* by the database, but the database is necessarily not your most flexible unit of management for storage. For example, consider the storage element shared by databases 1 and 2. If database 1 were migrated to a different SQL Server instance running on another server, perhaps its dedicated storage element could be migrated with it. The shared element, however, could not be so easily migrated because it's being used by *another* database.

This isn't to say that reconfiguring storage is insanely complicated, because of course it isn't. However, doing so can result in some downtime as well as some administrative time, and it's not entirely without risk in terms of data loss or additional downtime. All of these factors conspire to make storage configurations just a bit less flexible than they could be, meaning administrators try to avoid storage reconfiguration whenever possible, meaning you wind up with a lot of unused storage capacity floating around the infrastructure—capacity which is essentially locked to a specific instance, and which can't be easily used to relieve overcrowding on *another* instance.

Client Connectivity Requirements

The need for a SQL Server client application to know the name of a SQL Server computer and instance can also be a limiting factor in how dynamic the SQL Server infrastructure can be. Unfortunately, many application developers persist in hardcoding database connection strings, meaning that moving a database to another server—or even another instance on the same server—requires the application to be modified and redistributed (and the re-distribution is often the difficult part of that). Even with an application that is more reconfigurable, actually *performing* the reconfiguration can be difficult depending upon how the application was written.

Tricks exist that can help alleviate these issues. For example, one trick is to never allow clients to use actual server host names. Instead, a new DNS alias is created for each SQL Server instance that exists in the organization. The alias points to the physical server name that is hosting the instance. This allows each instance to have a unique “server name,” and if the instance is moved to another physical server, the only update needed is to the DNS alias record (typically a CNAME record). So a certain amount of flexibility exists for client connectivity if the organization has been smart. The task of moving an instance, from a client connectivity standpoint, can be accomplished fairly quickly by simply updating DNS.

 Note that this DNS technique doesn't abstract the *instance name*; DNS can't provide that service. Instead of accessing InstanceA on Server2, you set up DNS so that you're accessing InstanceA on ServerABC—with ServerABC simply “pointing” to Server2. If InstanceA needs to be moved to Server3, it can be, so long as it *keeps the same instance name*. Just re-point ServerABC to Server3, and the change is accomplished.


I'm not sure I'd qualify this as *flexible* within the Microsoft IOM definition of the word, though. DNS changes can take time to propagate through the organization, and Windows computers cache DNS information for about ten minutes, so you can't really rely on *instant* changes just using DNS. Other techniques exist, which I'll discuss later, but for now my point is this: SQL Server *itself* requires server names and provides very little in the way of flexibility for quickly changing server names. Other non-SQL Server techniques can be used to add flexibility to the infrastructure, but not all organizations are able to use all these techniques. Without using *some* kind of non-SQL Server technique, moving instances between physical servers can become anything from a hassle to an outright maintenance nightmare. That's one reason why administrators often loathe relocating SQL Server instances: Getting clients reconnected.

Disaster Recovery Requirements

The need for reliable disaster recovery is another factor that contributes to SQL Server sprawl and to a less-than-flexible infrastructure. Making backups of large amounts of data is one issue. For example, some organizations may rely on direct-attached tape drives to back up each SQL Server computer's databases. If the databases' size begins to exceed the capacity of the tape system—either in terms of storage capacity or the time in which it takes to back up the database—then backup becomes a constraint similar to memory or processor. Administrators respond by ensuring every server has excess capacity, creating a lot of excess, inflexible capacity within the infrastructure.

Many organizations today use much smarter and more flexible backup schemes. For example, they may have a centralized backup solution that backs up databases across a private network that is used only for backup traffic. This is a much more flexible scheme, as it centralizes excess capacity. Any server can benefit from any central excess backup capacity. Most organizations have a hybrid approach, using several centralized backup systems for various “clusters” of SQL Server computers. This helps to create a better balance between backup speed and excess capacity.

Business factors often step in to make the infrastructure less flexible, though. For example, consider Figure 2.5, which illustrates an organization's four geographically dispersed offices. Backing up SQL Server over a wide-area network (WAN) connection is rarely feasible, so each office has at least one backup solution in place for its SQL Server computers. In reality, political issues—departmental ownership of resources, for example—mean that each office has a couple of backup solutions, some shared by multiple servers and others dedicated to a single server. I've shown the backup solutions as colored bars underneath the servers that share each solution, with colors indicating the backup solutions' current utilization—red for almost maxed out, yellow for moderate utilization, and green for low utilization.

 Note that the utilization I'm showing refers to the backup solution, which as you can see is shared by different sets of servers at each location.

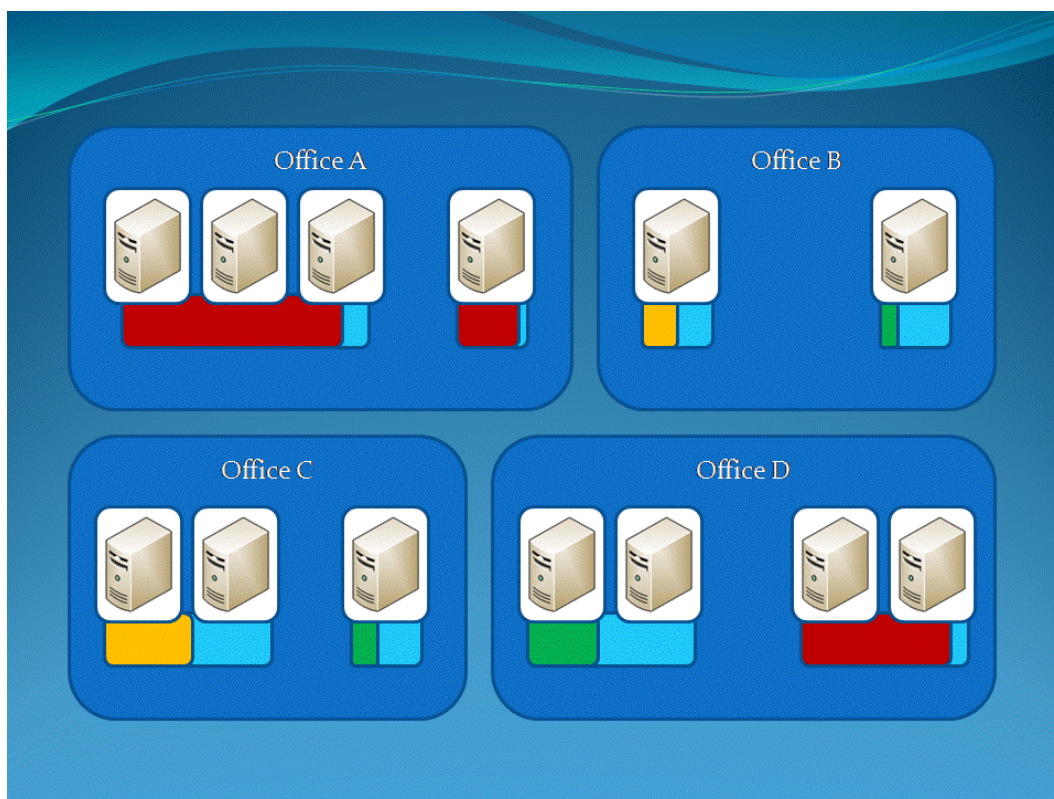


Figure 2.5: Backup solutions and capacities in use across a geographically distributed organization.

Once again, we see that ample excess capacity is present to accommodate the organization's total needs. However, this capacity isn't flexible. Office A, for example, is coming close to running out of backup capacity; the excess capacity in Office C, however, is of no help in solving the problem. Office B has more than double the capacity it requires, but that extra doesn't do Office D any good for their almost-overburdened solution. Even the excess capacity already available in Office D isn't available to the SQL Server computers that need it.

Often, the business factors that result in these situations are legitimate for other reasons—administrative control over resources, budgetary requirements, and so forth. However, it creates a hardware-centric infrastructure that is inherently less flexible than it could be. If Office A needed to implement a couple of new databases, they might well be looking at significant expenditure and effort to make that happen—even though the organization has already spent more than enough money on backup capacity. Although Office C could probably implement new databases without any issues, Office A has less flexibility than it really should.

Basic Server Consolidation

I want to start with a baseline for reducing SQL Server sprawl, so I'll start with the basic consolidation capabilities that are built into SQL Server itself.

Technology Overview

The *instance* is SQL Server's basic method for consolidation. Essentially, any server running SQL Server 2000 or later is ready to use this consolidation technique. Simply relocate each instance to a fewer overall number of physical servers. Of course, I say "simply" even though that relocation isn't always simple.

The SQL Server Connection

There aren't any built-in tools for "moving" a SQL Server instance from one physical server to another. Instead, you have to manually perform the task, which involves several distinct steps. You first decide whether you're going to create a new instance on the destination server or use an existing instance. Once you've decided what instance to use, you copy over the database. Data Transformation Services (called Integration Services in SQL Server 2005) can copy the database or you can simply copy the database files directly. This will involve downtime; you really need to take the original server offline so that changes aren't being made until the move is complete. Then you'll need to reconnect your client applications using whatever means those applications support and require.

This type of consolidation is almost entirely a manual process, and it can be time-consuming depending upon the amount of data involved. It does involve a certain amount of downtime, and it's not something you can do in the middle of the business day for production databases.

The Impact on Sprawl

This technique obviously has a positive impact on sprawl by reducing the number of servers that are being managed. As databases are migrated off of servers, those servers can be decommissioned and put to other, non-SQL Server uses or removed from the environment entirely.

The Impact on Flexibility

This technique doesn't have much of an impact on flexibility. You're certainly not going to be moving databases from server to server all the time, and organizations rarely consolidate in this manner for any reason other than reaction to a problem. For example, consider Figure 2.6.

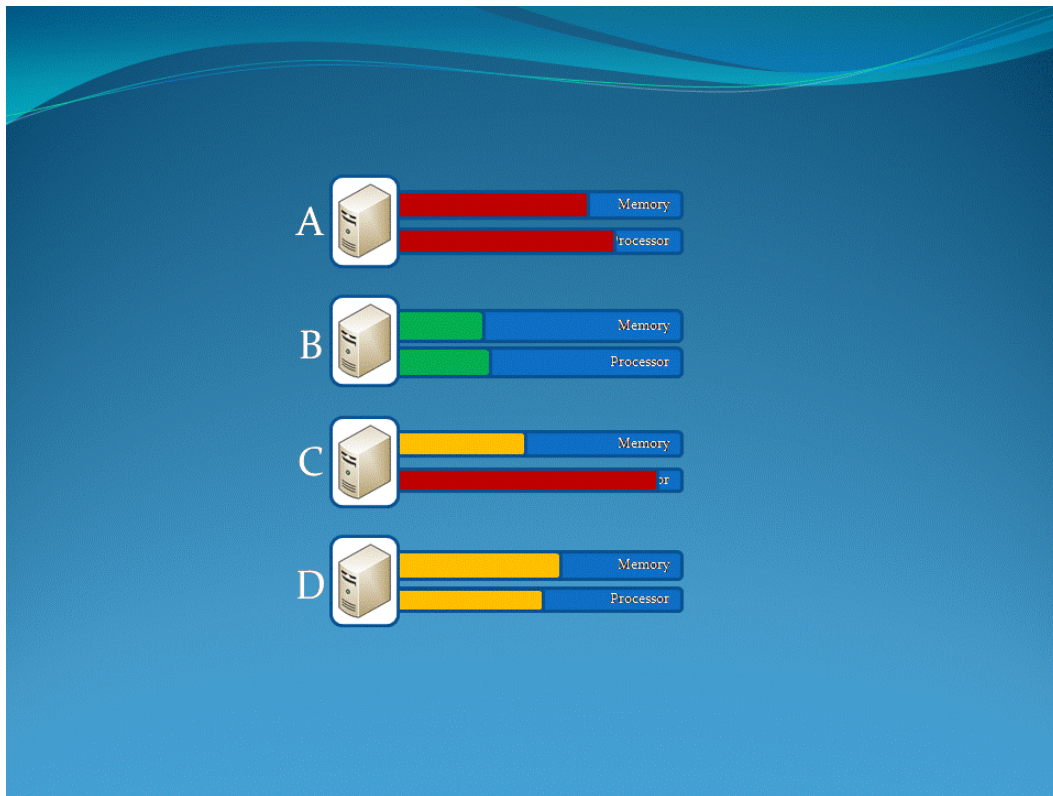


Figure 2.6: Lack of flexibility.

As shown, plenty of excess capacity exists across these four servers. In fact, you could probably combine Servers B and D. And let's say that a new database came along that would require about 60% of a server's capacity. Perhaps moving some workload from A to C would free up enough capacity on A to host the new database, without implementing a new server. Would the average SQL Server administrator do it? Probably not, because none of these decisions leaves much free capacity on any one server, and you can't really move databases and instances around easily enough to be doing it all the time. Consolidation using this technique is something you might do only in really obvious circumstances, such as when you have a couple of *very* under-utilized servers, and combining them would still leave plenty of free capacity.

Windows Server Clusters

Windows Server Clusters—that is, clusters built using the Windows Cluster Service—are primarily intended as high-availability solutions rather than a server-consolidation technique. However, they do seem to offer some interesting capabilities for server consolidation, as on the face of it they work around some of the issues involved with basic server consolidation. Let’s see how well-suited they are to reducing SQL Server sprawl and improving the flexibility of the infrastructure.

Technology Overview

Figure 2.7 illustrates a typical two-node cluster running SQL Server. Part of the clustering technology is the creation of virtual server names. In this case, SQLA, SQLB, and SQLC are all “computer names” that the cluster uses; only one cluster node—the “owner”—will respond to a given name at any given time. In this case, each virtual name maps to one of three SQL Server instances; as shown, two instances are running on the first node, and one is on the second node.

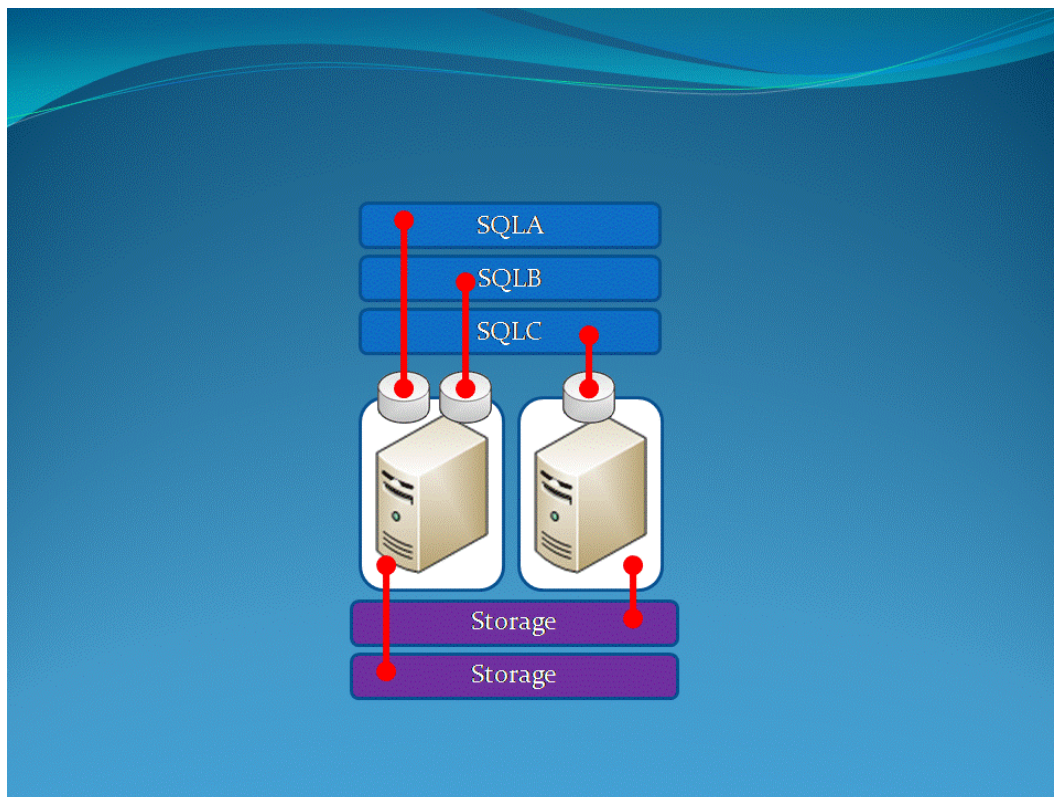


Figure 2.7: SQL Server in a Windows Cluster.

For each node that will be “active”—that is, for each node running a SQL Server instance that is fulfilling client requests—an external storage system is required. Therefore, this cluster requires two such systems. Each storage system is physically connected to each node (often via a SAN, although sometimes with SCSI cabling), although only one node “owns” a given storage system at any given time, and only the “owner” node can access a given storage system.

In a failure situation, the resources “owned” by the failed node are “seized” by the surviving node. That’s straightforward enough: If you can picture the second node in our cluster dying, then the first would “own” all three virtual server names, would run all three SQL Server instances, and would “see” both storage systems.

From an infrastructure flexibility standpoint, clustering is interesting because it allows SQL Server instances to “move” across nodes with relatively little effort. Each instance is actually installed on every node in the cluster; it’s just *running* on the node which “owns” that instance at the time. So if we needed to take the second node offline for maintenance, we’d just manually move its resources to node 1 for a while. This would seem to enhance server consolidation, as well. Since resources can be moved between cluster nodes, you might feel less need to have a lot of excess capacity. Any excess capacity on one node could be used for any workload, since workload could be transferred to that node to utilize what capacity it had available.

The SQL Server Connection

SQL Server actually needs only a minimal awareness that it’s running on a cluster in order to work properly. To oversimplify a bit, each instance of SQL Server is installed on each node in the cluster but is stopped. When a cluster “owns” an instance, it starts the instance and begins serving client requests.

Each instance usually *depends* upon other resources being owned by the same node. That is, if a node can’t “own” the external storage that holds an instance’s database files, and it can’t “own” the virtual server name that the instance uses to communicate, then the node can’t “own” the instance either. Those resources—storage, instance, and name—are transferred as a set. *Most* of the work is done by the Windows Cluster Service.

The Impact on Sprawl

On the face of it, Windows clusters might not seem to impact sprawl at all. After all, two servers are two servers, whether they’re in a cluster or not. The theory on positive sprawl impact depends upon the ability for the *cluster itself* to have the excess capacity you need to handle growth or spikes in workload; because resources can be transferred between nodes, you would perhaps only need one node with a lot of excess capacity—the others could run at closer to full capacity, meaning you could possibly eliminate some servers. Three- and four-way nodes add flexibility to this idea. The *actual* impact on sprawl might be minimal, though, because the impact on flexibility is smaller than you might think.

The Impact on Flexibility

Windows clustering isn't a free-for-all when it comes to moving workload between nodes. Consider our example cluster: Because the two instances running on the first node store their databases on the same external storage, both instances *depend* upon that one resource. Therefore, both instances must be "owned" by the same node at all times. If you move the SQLA instance to the second node, its storage (the lower one) would also have to move, which means the SQLB instance would also have to move. This is poor flexibility, since you can't be really granular when moving workload around. Of course, you could build your cluster to look more like Figure 2.8, with each instance having completely dedicated resources.

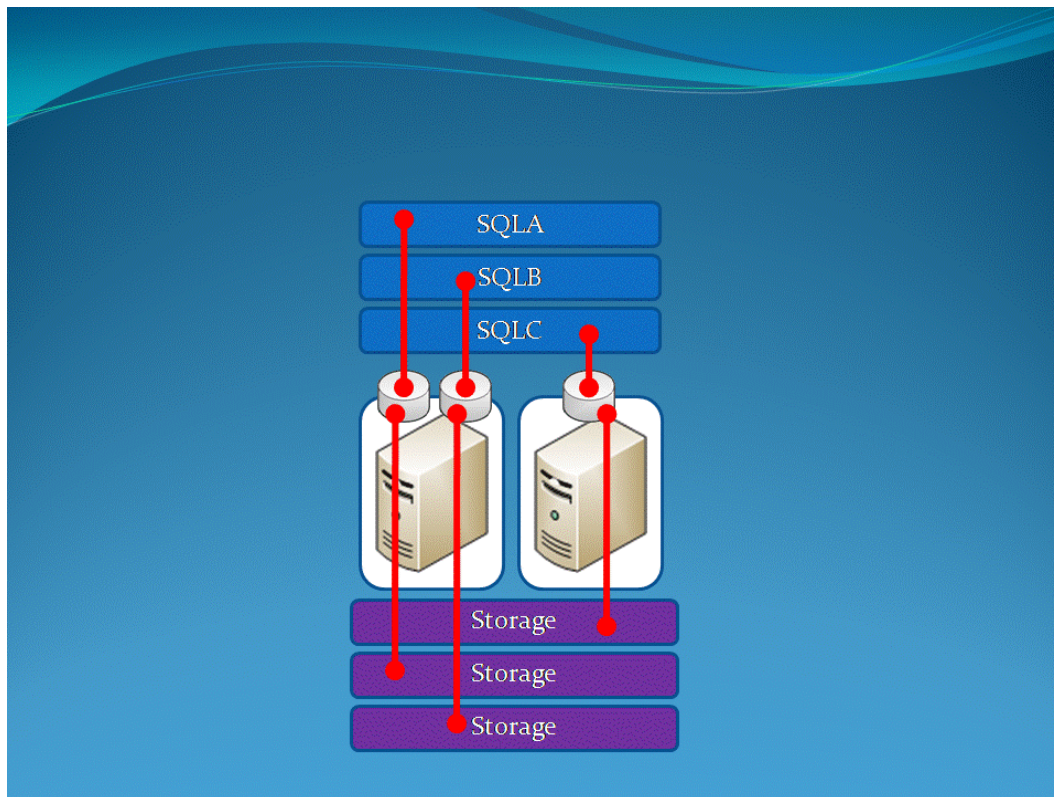


Figure 2.8: Dedicated resources for each instance in a cluster.

However, you now start to enter a realm of increasingly complicated resource management. Imagine a dozen instances running on this cluster, with a dozen virtual names and a dozen storage resources. Managing all of that becomes complicated, error-prone, and difficult. Simply configuring the storage resources to operate that way can require excessive hardware. For example, if those external storage systems were SCSI arrays, you might need a dedicated controller card for each (depending on the hardware you're using)—in *each node*. You'd quickly run out of expansion slots in your server! Even SAN-attached storage can present complications of a similar nature. Thus, although Windows clusters can add *some* flexibility to the infrastructure, they can't add *much*, simply because that's not the problem that Windows clusters were really designed to solve; Windows clusters are intended purely as a high-availability solution, and they simply weren't designed to address consolidation needs.

Hardware Virtualization

Technologies such as Microsoft Virtual Server and VMWare Server are becoming increasingly popular for server consolidation. Essentially, these technologies make it possible to run multiple “virtual servers” (commonly called *virtual machines* or VMs) on a single physical machine.

Technology Overview

As shown in Figure 2.9, virtualization essentially works by laying a “hypervisor” layer on top of the machine’s physical hardware. This layer is controlled by an operating system (OS) such as Windows.

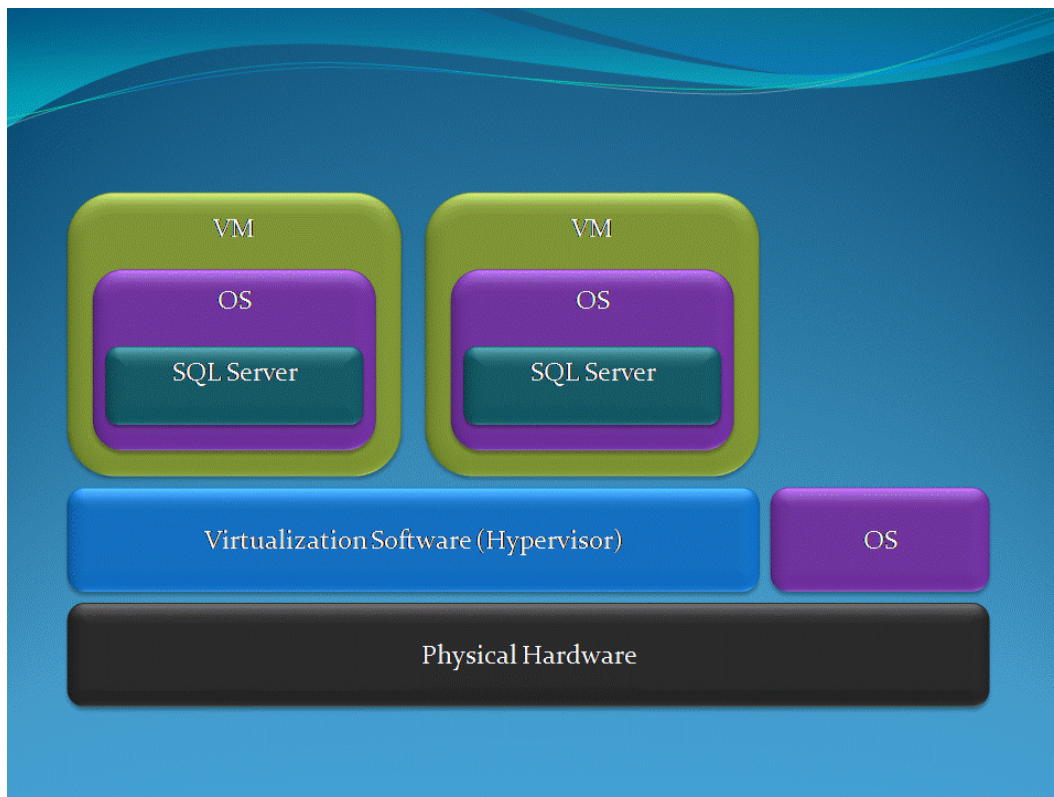


Figure 2.9: Hardware virtualization.



This model reflects a relatively modern way of building virtualization and is essentially how VMWare ESX Server is built; with software products like Microsoft Virtual Server 2005, or VMWare Server, the model is slightly different. With those products, the virtualization software runs *on top of* the host OS rather than *beside it*. This provides additional overhead for accessing the machine’s physical hardware, which can cause resource constraints.

On top of the virtualization software run VMs. These essentially emulate real physical hardware, and you install a real OS such as Windows onto them, and then install software such as SQL Server. Each VM has its own virtual hardware resources: memory, disk, processor, and so forth. “Within” the VM, the OS believes it is running on a dedicated physical machine. In reality, the VM’s virtual resources are mapped to the machine’s physical hardware resources, which are partitioned and shared across all running VMs.

The SQL Server Connection

SQL Server doesn't "care" about virtualization of this kind and doesn't need to do anything special to work with it. As far as SQL Server is concerned, the VM is a real, physical machine. It has a server name, memory, disk space, and everything else a physical machine would have.

The Impact on Sprawl

Virtualization can definitely have a positive *partial* impact on sprawl. By provisioning virtualization hosts as large, powerful servers, you can consolidate smaller machines—often older ones that use hardware that is inferior to the current state-of-the-art—onto fewer physical machines. Figure 2.10 illustrates the consolidation of five physical machines onto two virtual hosts.

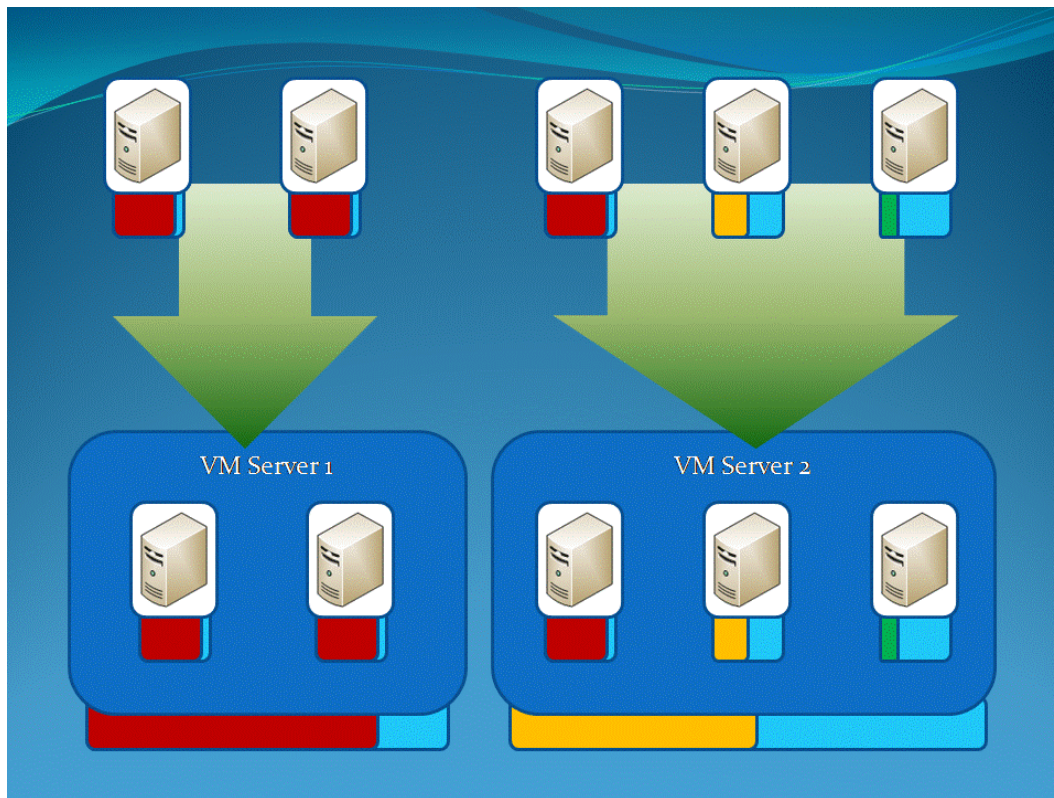


Figure 2.10: Consolidating machines with virtualization.

There's not exactly a formula for how many physical machines can go onto a single virtual host, though. For one, the virtual host may have more powerful processors, meaning each one is individually more powerful than a single processor in an older physical machine. And the overhead of the virtualization process itself is not insignificant, although it can be difficult to measure accurately.

Also consider that *sprawl* isn't just the physical count of computers. Although reducing computer count saves data center space, converting physical machines to virtual machines doesn't lessen the amount of OS maintenance you have to perform, the amount of SQL Server maintenance, and so forth. In fact, maintenance *increases*: Not only must you keep the OS and software on each VM up-to-date; you also have to worry about the OS and software on the physical machine that all the VMs run on. In this regard, virtualization has a *negative* impact on sprawl, in that it can slightly increase the amount of computers, both physical and virtual combined, that you have to maintain.

The Impact on Flexibility

Virtualization can also seem to have a positive impact on flexibility. Most high-end virtualization platforms, such as VMWare's ESX Server, provide tools to help move virtual machines between physical hosts to quickly rebalance workload as needed. With this technique, the virtual hosts—the physical machines—become one giant pool of resources, and workload—running inside VMs—can be spread across them fairly dynamically. Tools vary in their ability to seamlessly move VMs; none can do so without at least minor downtime, and few can do so without fairly major downtime.

In addition, because SQL Server itself is such a big consumer of computer resources, the additional overhead of virtualization can be something of a waste. Virtualization is absolutely ideal for combining disparate applications running on disparate OSs; it's perhaps a good, but not great, solution for SQL Server consolidation, at least for low-use databases. However, virtualization solutions simply aren't appropriate for high-use databases—they just can't provide sufficient performance. Keep in mind that SQL Server, through its instances, already has a rough form of “virtualization;” what's needed is a way to quickly move instances across physical hosts, without having to encapsulate those instances in a whole VM—along with a complete OS to run on.

Pooled-Resource Clustering

Pooled-resource clustering is a combination, of sorts, of Windows clustering and virtualization. Essentially, it's a means of allowing SQL Server instances to become “virtual SQL Servers” in the sense of true VMs. They're encapsulated and can be easily and quickly moved between physical computers. It borrows concepts from Windows clustering but does so with a crucial difference that eliminates the resource management overhead that comes along with Windows clustering. Another term for this technique is *shared storage* clustering.

Technology Overview

With pooled-resource clustering, a cluster of servers—which can all have different hardware specifications—are connected to a single, shared storage system (typically a SAN). By leveraging a Windows-based cluster file system, *each* of the nodes in the cluster can “see” the same storage area. That is, they all have (for example) an S drive, and all of the files on the entire storage system are exposed to each node through that S drive.

Each server runs one or more instances of SQL Server. In reality, each instance may be *installed* on each server, although it is only *started* on one node at a time—much like Windows clustering. Like Windows clustering, a virtual server name is usually associated with each instance, and is “owned” by the node that “owns” the corresponding instance. Figure 2.11 illustrates the basic configuration.

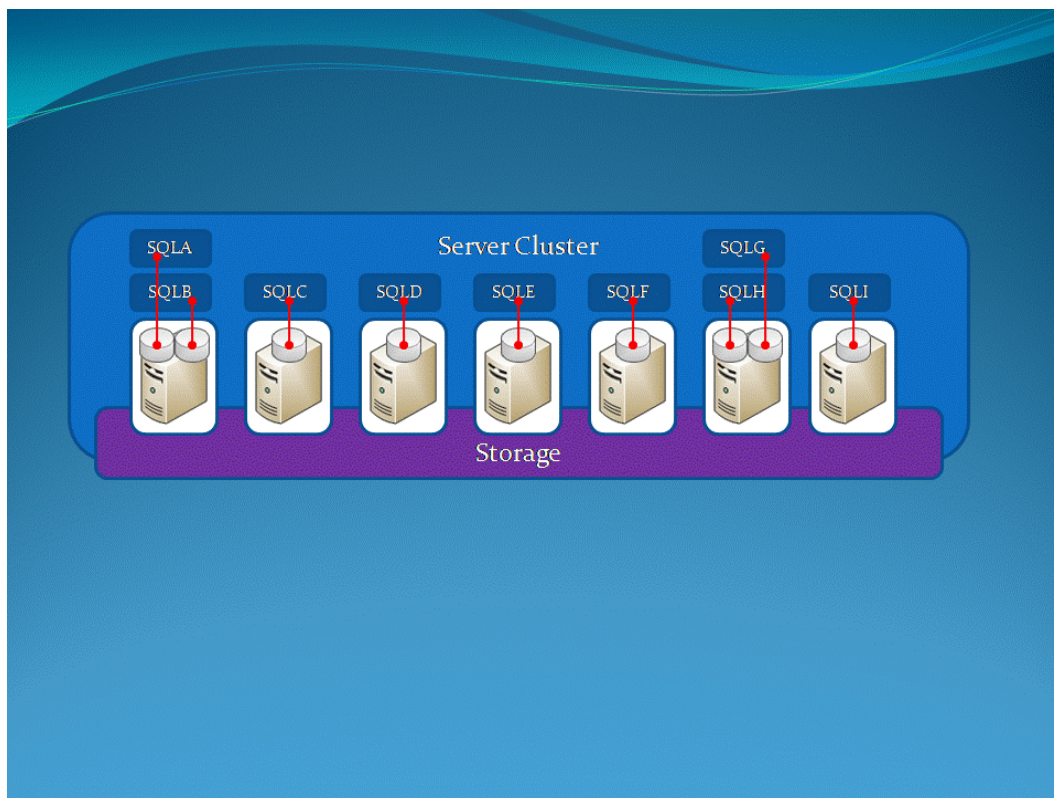


Figure 2.11: Pooled-resource clustering.

The trick with this technology is that *no node must try and access the same database files at the same time*. The reason is that Windows always assumes that all local storage is private and contains no mechanism for resolving access conflicts. Fortunately, each instance of SQL Server would be opening different database files, and would be locking them when doing so. For example, the “SQLA” instance might be installed on all seven nodes but only running on node 1. If node 2 tried to start its “SQLA” instance, the instance would find that the database files it was configured to use would be *visible*, but not *accessible*, because the instance running on node 1 had them locked. The instance would simply shut down, being unable to open its configuration database or any other databases. Figure 2.12 illustrates this activity.

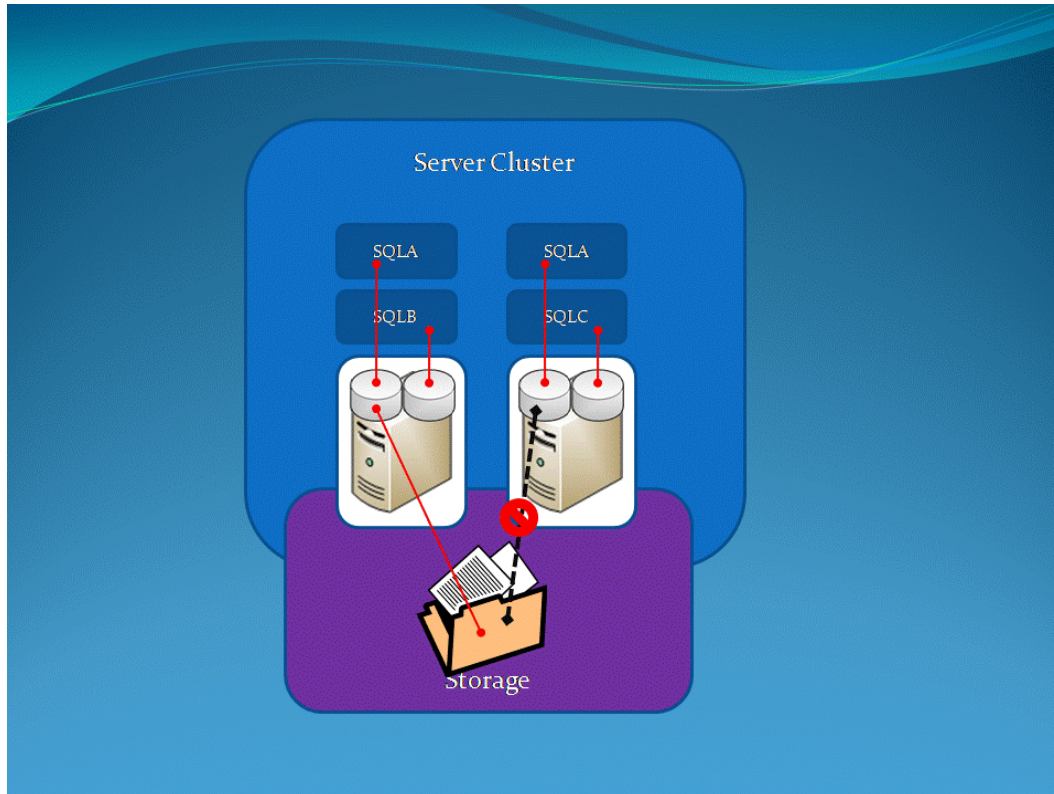


Figure 2.12: File locking prevents duplicate instances from accessing the same files.

To “move” the SQLA instance to the second node simply requires shutting down the instance on node 1 and starting it on node 2—which, since node 1 is no longer running, will be able to access the database files on the shared storage area.

Automating this shutdown and startup process provides a very quick and robust high-availability solution. With this shared file system, data ownership doesn’t have to be transferred; it’s already there. Thus, clusters can have *many* servers providing active-to-active failover support for each other.

Each cluster of physical nodes only requires a *single* storage area; instances depend upon their *files* rather than the storage area itself, so each instance of SQL Server becomes independent. Further, your configuration can be as flexible as you like: The “SQLA” instance, for example, might be capable of running on four out of the seven nodes, if those were the only nodes with sufficient hardware resources to support that instance.

The SQL Server Connection

SQL Server itself doesn't need to do anything special to run in this kind of environment; in fact, it won't "know" that it *is* running in this kind of environment. External cluster management software is responsible for starting and stopping SQL Server instances as necessary, either on demand or in response to a hardware failure in a given node.

The Impact on Sprawl

The impact on sprawl is positive with this technique. Every server can be utilized to its fullest. As shown in Figure 2.13, you can balance the workload however you want. In this example, the second server has plenty of excess capacity. If another server becomes overworked, that excess capacity can be *used* by transferring workload to the less-busy server. *Every* server doesn't require excess capacity, and so you can have fewer computers to handle the same workload that you have today.

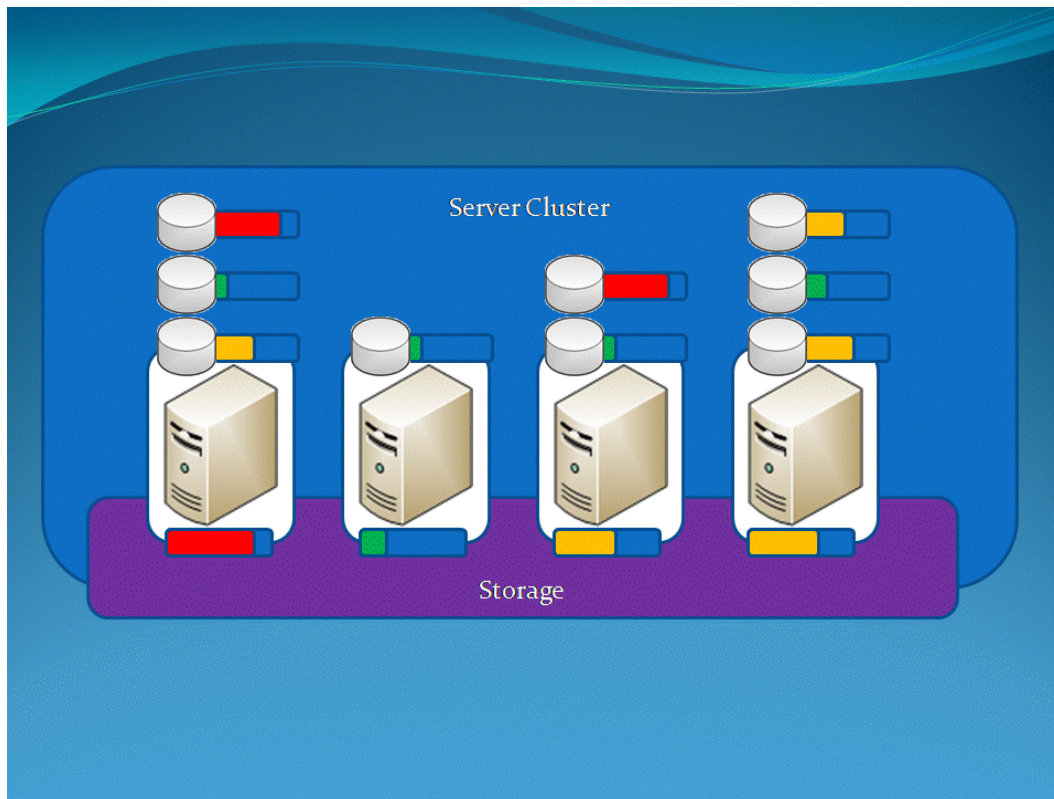



Figure 2.13: Spreading workload across the pool.

 In this figure, I'm showing server capacity and workload in colored bars below each server. The bars next to each database represent that database's activity level.

This is a *true* reduction in sprawl, with a commensurate reduction in maintenance overhead for maintaining OSs and software. Further, when your pool itself eventually reaches its maximum capacity, you can add another, brand-new, leading-edge server to the pool. That server will likely be able to replace perhaps two older servers from the pool, while still having excess capacity for overall workload growth, which helps to *maintain* the reduction in sprawl.

The Impact on Flexibility

This technique has the best impact on flexibility of any I've discussed so far. Because all servers can see each instance's database, moving an instance to another server is simply a process of shutting off one instance and starting another—a process that takes about 30 seconds. Need to implement a new database? Put it on whatever server has the capacity for it. If it'll initially be a low-use database, put it on a server like the first node in Figure 2.13; if the workload goes up over time, simply relocate it in seconds to the third or fourth node. You're still maintaining extra capacity in the infrastructure, but that capacity is now available to *any SQL Server instance that needs it*, not just the instances installed on a particular computer. Want to completely upgrade all of your server hardware? Fine: Add new servers to the pool, move workload to them, and turn the old servers off. Done. Do you desperately need one of your SQL Server computers for another task for a short period of time (perhaps to replace a failed server that was doing some other task)? Relocate your SQL Server workload to other servers in the pool, and take the server you need out of the pool. You get infinite flexibility, on demand. Rearranging your infrastructure is no longer a hassle or potential risk: The infrastructure is *specifically designed for change*. In Microsoft IOM terms, it's *dynamic*.

Infrastructure Optimization: The Positive Impact on SQL Server Sprawl

With an optimized infrastructure, you'll definitely reduce SQL Server sprawl but you'll also give "new life" to your business capabilities by being more flexible and more dynamic in the use of your infrastructure's resources. I want to point out that the techniques I've shared in this chapter—particularly pooled-resource clustering—probably still meet the political needs of your organization. Remember that each instance of SQL Server is an "island," with its own server-wide configuration. You can set up each instance to serve whatever business or political needs you have; if a department needs "total and exclusive control" of their database, fine: Give them an instance. *You* decide what hardware it runs on. SQL Server's internal security is robust and granular enough to allow for any combination of scenarios.

Coming Up Next...

In this chapter, I looked at various ways of consolidating SQL Server computers to reduce sprawl and explored each technique's impact on the infrastructure's flexibility. My conclusion is that pooled-resource clustering, of the techniques I've discussed, provides one of the most positive impacts on sprawl and on the infrastructure itself. By creating the ability to pool processor, memory, storage, and other resources, and giving you the ability to almost instantly relocate workloads across that pool, you create a highly dynamic infrastructure for SQL Server. In the next chapter, I'll focus entirely on this pooled-resource clustering idea, looking at some of the base technologies that make it possible, and looking closely at how to actually build this type of infrastructure into your environment.

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>.