


Realtime
publishers

"Leading the Conversation"

The Shortcut Guide[™] To



SQL Server Infrastructure Optimization

sponsored by



i n v e n t

Don Jones

Introduction to Realtimepublishers

by Don Jones, Series Editor

For several years, now, Realtime has produced dozens and dozens of high-quality books that just happen to be delivered in electronic format—at no cost to you, the reader. We’ve made this unique publishing model work through the generous support and cooperation of our sponsors, who agree to bear each book’s production expenses for the benefit of our readers.

Although we’ve always offered our publications to you for free, don’t think for a moment that quality is anything less than our top priority. My job is to make sure that our books are as good as—and in most cases better than—any printed book that would cost you \$40 or more. Our electronic publishing model offers several advantages over printed books: You receive chapters literally as fast as our authors produce them (hence the “realtime” aspect of our model), and we can update chapters to reflect the latest changes in technology.

I want to point out that our books are by no means paid advertisements or white papers. We’re an independent publishing company, and an important aspect of my job is to make sure that our authors are free to voice their expertise and opinions without reservation or restriction. We maintain complete editorial control of our publications, and I’m proud that we’ve produced so many quality books over the past years.

I want to extend an invitation to visit us at <http://nexus.realtimepublishers.com>, especially if you’ve received this publication from a friend or colleague. We have a wide variety of additional books on a range of topics, and you’re sure to find something that’s of interest to you—and it won’t cost you a thing. We hope you’ll continue to come to Realtime for your educational needs far into the future.

Until then, enjoy.

Don Jones

Introduction to Realtimepublishers.....	i
Chapter 1: Traditional Challenges and Their Impact.....	1
The Symptoms of an Un-Optimized Infrastructure	2
Isolated Systems and Data Silos	2
Reactive Management.....	3
Availability Concerns	3
Top-Level Causes of the Un-Optimized Infrastructure	4
Technological Causes	4
Performance Constraints.....	4
Storage Constraints	5
Chassis Constraints	5
Business Causes	5
Branch Offices	6
Departmental Servers.....	6
Project-Based Servers	6
Top-Level Causes: A Summary.....	6
The Result of an Un-Optimized Infrastructure	7
Maintenance Overhead	7
Application and Data Availability	8
Management Overhead.....	10
Disaster Recovery Overhead.....	10
Life Cycle Management.....	11
Storage Management	11
Performance Management	11
Security and Auditing	11
Introducing the Infrastructure Optimization Model.....	12
Basic.....	13
Standardized.....	13
Advanced	14
Dynamic	14
Application Platform Optimization.....	15
Keys for SQL Server Infrastructure Optimization.....	16
Abstracted	16

Segmentation.....17

Manageability18

Performance18

Availability18

Life Cycle.....19

Disaster Recovery19

Dynamic19

Coming Up.....19

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.

Chapter 1: Traditional Challenges and Their Impact

SQL Server can be an organization's greatest asset and—from a corporate management perspective—a complex challenge. The reason is that SQL Server (or any enterprise-class database system, for that matter) places a heavy demand on infrastructure resources such as servers, the network, and storage. Many organizations continually struggle to scale their SQL Server installations to meet business demand—including requirements for availability in the face of hardware failure—while at the same time bemoaning the “data center bloat” that seems to be the unavoidable companion of large SQL Server installations. A lot of that is simply the cost of SQL Server's success in the enterprise: As companies find more uses for SQL Server, there are inevitably more SQL Server installations to deal with. Some of the challenge comes from what I'll call *traditional SQL Server architecture*, which too often consists primarily of double-clicking Setup, or architecting SQL Server for performance without taking manageability into account.

In fact, SQL Server and “data center bloat” don't necessarily go together any more than SQL Server and “difficult to manage” do. With the right tools and techniques, you can have a top-performing SQL Server infrastructure *without* having to cram your data centers with so much hardware that they're all but overflowing. Some of these tools and techniques may not seem very obvious, which is perhaps why many SQL Server architects don't discover them right away. They are, however, extremely effective.

It's all about *infrastructure optimization*, or letting your application get the very most from the infrastructure that it's running on. Before I can begin to share some of these new techniques, however, I need to back up a bit and explain exactly *why* so many organizations aren't really optimizing their SQL Server platform.

The Symptoms of an Un-Optimized Infrastructure

I want to stress that the “management problems” most often associated with SQL Server are really associated with the underlying infrastructure: too many servers, too many disparate storage devices, and so forth. When the infrastructure that SQL Server relies upon isn’t optimized, what sort of “big picture” problems can you expect to encounter?

Isolated Systems and Data Silos

One sure sign of an un-optimized system is *data silos*, as illustrated in Figure 1.1. This occurs when each major application within a company is not only given its own database—as you would expect—or its own SQL Server instance—which might make sense—but its own *SQL Server computer*—sometimes its own *server cluster*.

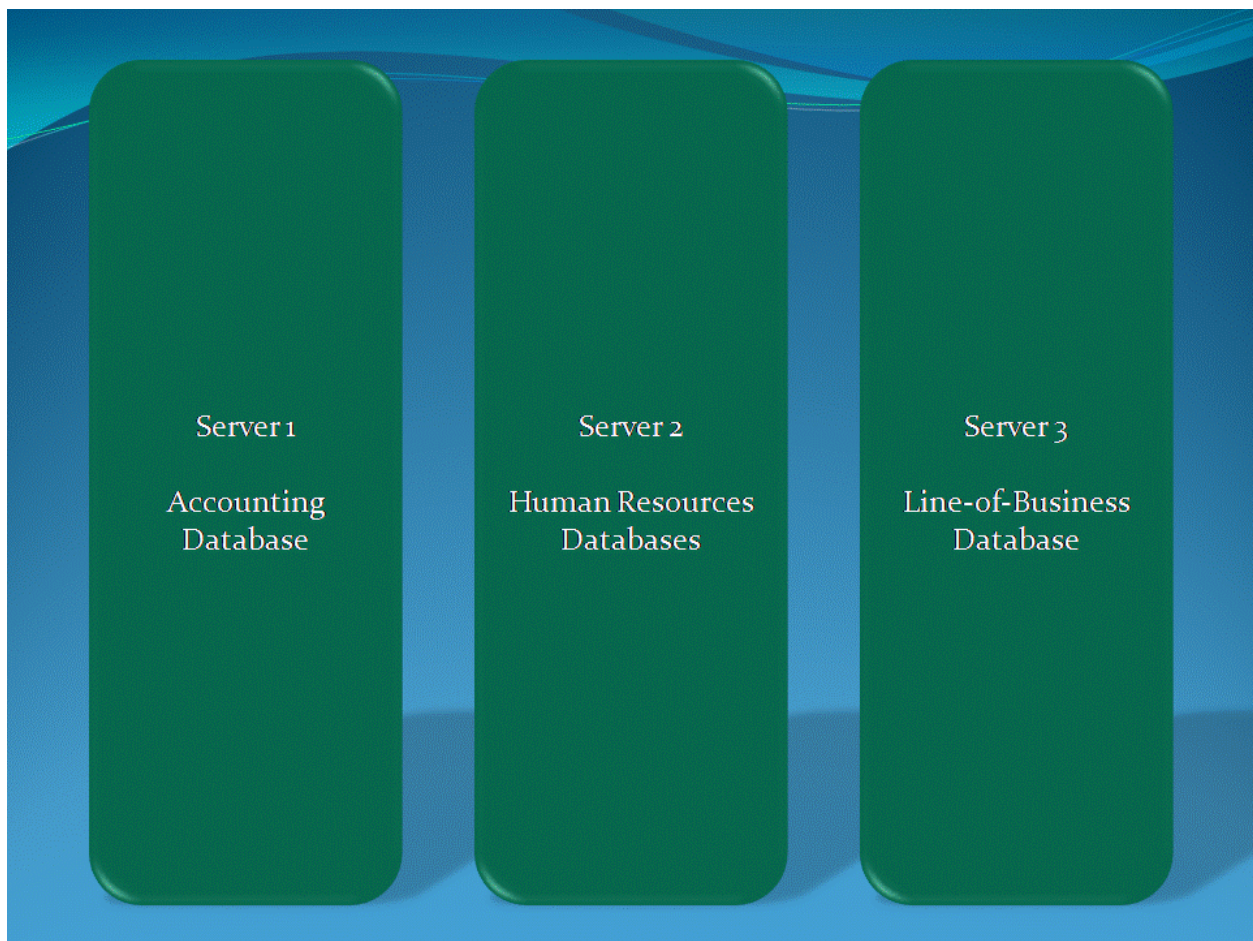


Figure 1.1: Data silos result in hardware bloat and are a symptom of an un-optimized infrastructure.

Some IT administrators would argue that these single-purpose servers are the *cause* of an un-optimized infrastructure—and, to a point, I suppose I would agree. Other administrators would tell you that this is an *optimal* architecture because they like this kind of isolation between applications—even if none of these servers are even close to fully utilized. However, I think that these types of isolated systems are for the most part a *symptom* of an un-optimized infrastructure. I'll explain why.

Imagine you work for a department in your company that needs to deploy a SQL Server computer for a project, or perhaps a department-specific application. You're perfectly willing to have the database hosted on your company's "big" SQL Server—the one that perhaps runs SAP, or Peoplesoft, or some other large enterprise application. The database administrators (DBAs) who "own" that server won't hear of it, though, because you'll be occupying precious "growth capacity" on their machine—unused resources, including computing power and storage, that they're saving against future need. So you go out and pay for your own SQL Server computer. It's probably housed in the corporate data center and may even be tended to by those same DBAs, but it's a machine dedicated to you. Most likely, a lot of your machine's resources are being wasted, too, simply because the machine is too big for you one little database.

So the *symptom* is data center bloat but the *cause* is that the infrastructure isn't optimized to the point at which your DBAs feel comfortable using every last iota of resources available to them before adding more—they insist on maintaining a degree of wasted, unused resources on each SQL Server machine, "just in case." It's not their fault, of course—it's the infrastructure's fault.

Reactive Management

Reactive management is another symptom—not cause—of an un-optimized infrastructure. Reactive management simply means managing the latest crisis: When Server "X" runs out of capacity, everyone runs around trying to fix the problem, implement a bigger server, or some other solution. Although corporate management culture can certainly take some of the blame for this style of management, for the most part, the blame can be laid squarely on the infrastructure. If it were capable of adapting more quickly and easily to changing business needs, then new business needs could be handled smoothly, without all the running around and firefighting.

Availability Concerns

One last major symptom—not cause—of an un-optimized infrastructure is availability concerns. In most cases, organizations would prefer that their databases never go down, even in the face of scheduled maintenance. Without an optimized infrastructure—one capable of moving database workload around to any available computer, at any time, without interruption—the only way to achieve that type of availability is through relatively simplistic, resource-dependent clustering mechanisms, such as Windows Cluster Service. Although those kinds of clustering solve the unplanned downtime problem, they also create *more* data center bloat by doubling (or more) the number of servers, storage resources, and other infrastructure elements. Think about it this way: If you have a two-node SQL Server cluster serving a single application, and the cluster's "active" node is at 30% utilization, the overall cluster is at just 15% utilization—keep in mind that there is a "passive" Server B essentially doing nothing! That's the type of utilization that provides great fault-tolerance but at a pretty high cost in terms of hardware and ongoing maintenance. Again, this is a *symptom*; if the infrastructure were properly optimized, less of that extra hardware would be *necessary*.

Top-Level Causes of the Un-Optimized Infrastructure

So where does an un-optimized infrastructure come from? It's difficult to find all the root causes because they seem so embedded in the way we're used to working. Often these bad habits were derived from working with the product and institutionalizing workarounds for the product's deficiencies. Even when the deficiency is addressed, the workaround persists. However, there are clear technological and business causes that *result* in an un-optimized SQL Server infrastructure. Many of these causes are part and parcel of traditional architectural techniques that have simply met their limit in terms of business agility; by examining some of these causes, we'll understand what we need to change about our infrastructure architecture in order to achieve some degree of optimization.

Technological Causes

Technological causes are typically not bad decisions or even bad products (although there are obviously exceptions). Instead, they're often just limitations of the way we look at infrastructure architecture.

Performance Constraints

Traditional thinking tells us that the only way to improve SQL Server performance is to buy a bigger server. SQL Server is almost always the first application to benefit from new processors—some of the first multi-processor, multi-core x64 servers, for example, were sold as SQL Server computers. And in many cases, “bigger is better” is still true for SQL Server. It's pretty rare, though, to find even massive enterprise applications that can't be satisfied with a 4-way dual-core 64-bit server with lots of memory installed.

However, in reality very few databases require everything that a single server can deliver; in many cases, several of an organization's databases could be “stacked” onto a single powerful server. But the server has always been held to be the performance constraint for SQL Server—scaling up with bigger hardware, in other words, can be better than scaling out to multiple servers through the use of a distributed database. Any given server can only be expected to support a finite amount of SQL Server workload, so you tend to buy a server capable of handling all the workload you think you'll *ever* get, and then you just put a single task on that server—“reserving” the rest of its capacity for growth. This is the architectural technique that often first leads to data silos and data center bloat.

Storage Constraints

Traditional architecture assigns storage space to a single computer. The reasons are fairly obvious; the concept of multiple servers sharing the same storage space is technologically challenging. In many cases, the storage assigned to a given server is pulled from a larger pool of storage available from a Storage Area Network (SAN); although the SAN is one big “cloud” of storage, individual chunks are carved out and assigned to specific computers and aren’t typically shared between them.

This technique of “assigning” storage plays a crucial role in SQL Server infrastructure. SQL Server, of course, needs lots and lots of storage. Once a “chunk” is assigned to a given computer, it may or may not be straightforward to resize that “chunk” in the future; organizations tend to assign the maximum space they expect SQL Server to need for a given database application. Assigning additional “chunks” in the future, to accommodate long-term growth, is always an option but can become expensive both in terms of hardware and in terms of resource management.

Chassis Constraints

Servers can, of course, be upgraded—but only to a point. Most mainstream servers support a maximum of four to eight processors and are typically locked into a single family of processors. Servers can only accommodate so much memory; even large x64 servers have an upper limit on how much memory can be physically installed in the chassis. These are hard limits; you can’t “squeeze” an extra processor under the hood, so once you’ve upgraded a server’s hardware to its maximum capability, you’re done. If the SQL Server application running on that server needs more hardware resources, you’re faced with the sometimes-ugly task of migrating the application to an all-new server.

That is why SQL Server computers typically have a lot of extra capacity: Migrating databases from one server to another is painful, but once you reach the limits of your hardware, you have no other options. Thus, it would seem to make sense to build servers with plenty of “free space” or “room to grow” and not load them up to maximum capacity right at the outset.

Business Causes

Not all causes of un-optimized SQL Server infrastructure can be laid at the feet of technology. In some instances, business requirements force architects into less-than-optimal designs. Again, I have to stress that these business requirements *aren’t bad* in and of themselves; it’s just that—using traditional architecture—they often lead to an un-optimized infrastructure.

Branch Offices

Branch offices are one of the most difficult aspects of any IT infrastructure, and it seems as if software vendors never take branch offices into consideration when designing their products. Branch offices typically need high-speed—for example, local—connectivity to certain resources, practically demanding their own on-site servers. Those servers are rarely fully utilized because many branch offices are (almost by definition) relatively small. Branch offices may or may not have their own dedicated IT administration resources, meaning these widely distributed resources (servers, storage, operating systems—OSs, and so forth) may need to be centrally managed—imposing a higher level of overhead for the remote administrators.

Branch offices may also be a cause for data center bloat. Branch offices may require dedicated databases, meaning they often wind up with those databases on a dedicated server. Again, this server is rarely fully utilized, creating wasted resources in addition to the distributed management overhead.

Departmental Servers

As I've already discussed, individual departments often have specific applications that require their own database, which in many organizations leads to the department “owning” (if not actually administering, although sometimes that's the case, too) their own dedicated SQL Server computers. After all, companies and departments have found SQL Server to be easy to deploy, relatively inexpensive, easy to administer, and so forth—it's easy and inexpensive to just spin up a new database server when you need one for a project, or a department, or any other need. However, this creates “bloat,” meaning the organization winds up having more SQL Server computers than it really needs based on the total SQL Server workload being served.

Project-Based Servers

Project-specific database servers—whether “owned” by a department or used in a more cross-organization fashion—are also one result of traditional SQL Server architectural decisions, and a cause of bloat. Project-based servers are amongst the worst kind of bloat because they're typically designed to be used for a relatively finite period of time; a design decision that often leads to the use of older, “spare” hardware. Too often, the project extends to near-infinite duration, meaning that older hardware continues to be a management and maintenance burden. And, as with most bloat, the server is rarely well-utilized—creating additional capacity in an infrastructure that rarely requires it.

Top-Level Causes: A Summary

It's important to remember that none of the technological or business causes I've discussed are inherently bad. Desiring department-specific databases, or project-specific databases, is perfectly normal and acceptable. Wanting to maintain “room for growth” in the database infrastructure is absolutely necessary. The problems come from technological limitations—such as chassis constraints—and from *perceptions* of how SQL Server must be designed in order to meet business requirements.

Simply put, the design choice of, “let's use a dedicated server for that” happens too frequently, when it isn't actually necessary in order to meet all the organization's needs and goals. Dedicated servers, however, are the cause of bloat—one aspect of an un-optimized infrastructure. What are some other results of an un-optimized infrastructure?

The Result of an Un-Optimized Infrastructure

Let's quickly define *un-optimized infrastructure* to be sure we're on the same page with the impact of having one: *infrastructure*, of course, refers to the underlying, supporting technologies that make something—in this case, SQL Server—work. SQL Server's infrastructure consists of obvious elements such as servers, storage resources, backup and restore resources, networking components, and so forth. Less-obvious elements include data center resources, power requirements, administrators' time, and so on. *Un-optimized* simply means that the infrastructure elements involved aren't being used to the best effect or maximum capability or capacity. In other words, the infrastructure *as a whole* has a lot of wasted or unused resources in it.

“Extra” isn't always a good thing: Imagine having a lot of extra food in your kitchen, for example. It takes up room, spoils if it goes unused for too long, higher purchase cost than necessary, and has other negatives. “Extra” money in your bank account might seem nice, but if it's just sitting there earning no interest, then it's not really working to its maximum effect for you. “Extra” insurance might give you a feeling of comfort and security, but it costs a lot and if it's never used, not only is that “extra” capacity wasted but so is all the money you spent acquiring and maintaining it.

As I'll discuss later, one goal of an optimized infrastructure is to achieve agility—the ability to quickly rearrange IT resources to smoothly meet changing business needs. An optimized infrastructure isn't the only way to achieve agility. You could simply spend a fortune on extra infrastructure resources, for example. However, the cost of achieving agility that way is prohibitively high, which is why few organizations have done so. An optimized infrastructure both saves you money *and* provides agility. But back to the topic at hand: If you've got an unoptimized infrastructure, what are some other negative effects you can expect it to deliver?

Maintenance Overhead

If *un-optimized* translates loosely to “too many extra resources,” an increase in maintenance overhead is probably an obvious negative impact. Server management—not to mention storage management and management of other types of resources—isn't a linear thing. Managing 10 servers might require a couple of hours per week apiece; managing 20 might require three. That's because, as the number of resources increase, certain overhead elements—inventory, SQL Server patch management, Windows OS patch management, hardware failure management, warranty management, and so forth—simply become more cumbersome. Figure 1.2 illustrates the curve that I've experienced in the past, with the number of hours expended each week increasing almost logarithmically with the number of resource elements that are being managed.

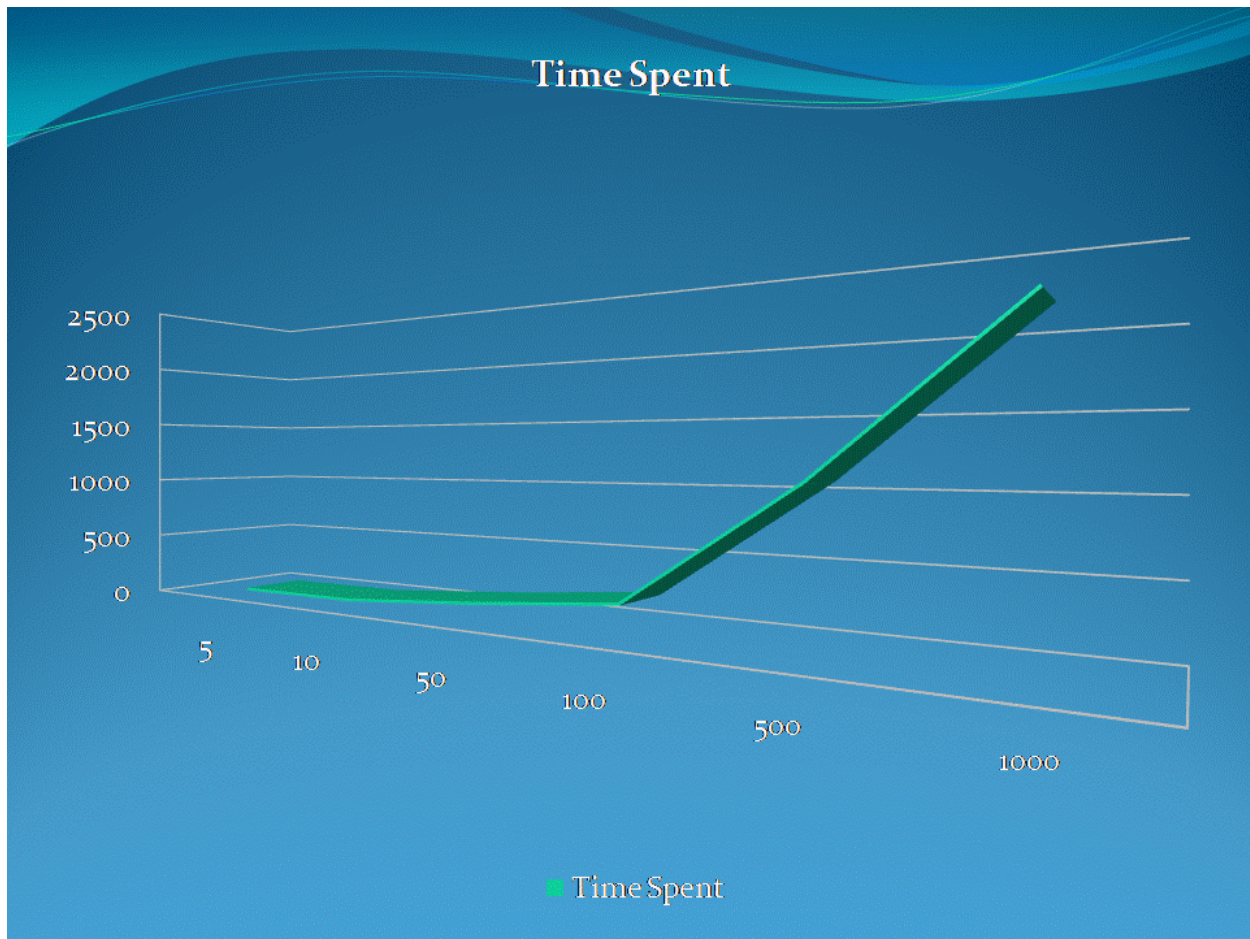


Figure 1.2: Maintenance overhead increases significantly as resources are added to the infrastructure.

Although it might initially seem like a good idea to have extra resources in case they're needed, there is a very real cost in terms of maintenance that is associated with such resources.

The problem gets worse when SQL Server is being independently deployed by individual departments. Too often, they create their own management processes, resulting in inconsistency, more difficult auditing procedures, different patch levels, and more—exponentially increasing the effort spent within the organization to maintain and manage SQL Server.

Application and Data Availability

An un-optimized infrastructure has a significant impact on application availability, as well. Some companies feel that hardware clustering is an ideal way to increase application availability. After all, the odds of two entire servers (or more, in larger clusters) utterly failing are pretty remote. But *it can* happen, and so some companies add even *more* resources to create a “never fails” infrastructure. That is an infrastructure positively laden with wasted, difficult-to-manage resources.

There is a vicious cycle here, too. More resources provide better availability but increase maintenance and management. An increase in maintenance and management can lead to improper maintenance and management, which can negatively impact availability. Figure 1.3 illustrates this seemingly endless circle.



Figure 1.3: The need for better availability leads to adding resources, which leads to increased maintenance (with a higher risk of mistakes), which negatively impacts availability.

Ideally, availability is something an optimized infrastructure *gives you for free* without the need for additional “availability-dedicated” resources and without the companion rise in maintenance costs and complexity that comes with adding resources to the infrastructure.

Management Overhead

An un-optimized infrastructure has a particularly chilling effect on management. One aspect of this stems from the number of resources being managed. There are simply too many, in most cases, to manage effectively. Management loses track of the number of resources present on the network, often forgetting about “extra” capacity and instead implementing *more* resources when additional projects kick off.

Another management impact comes when the business needs a change in the infrastructure. Perhaps the organization made a major acquisition, launched a new product division, or changed the way it does business; whatever the cause, the business needs have changed. Un-optimized infrastructures, however, don’t readily *accept* change, and so every change in the business results in a management headache for the infrastructure: Resources are shuffled, redeployed, migrated, upgraded, and changed, all with incredible manual labor. Manual labor of course means risk of error, and so more management resources are spent analyzing risks and coming up with mitigation plans. More management time is spent tracking the project, and when it’s all over, often as not, the business needs have changed *again*.

This is such an issue in most organizations that many of them simply avoid changing business needs. In other words, they let the inflexibility of their own infrastructure dictate what the business can do, and in today’s increasingly competitive marketplace, that’s simply not acceptable.

Imagine a world where you discover that a new business acquisition will put a specific SQL Server over-capacity. No problem: You have another server with plenty of capacity that is only handling a few file-serving tasks. You push a button, and SQL Server is installed on that server. The over-capacity database is instantly and automatically moved to the new server, and users never realize the move took place—it happened in seconds. Other databases move onto the now-empty database server, filling it almost to capacity. Those databases leave older server hardware vacant, allowing you to dispose of legacy hardware that may be becoming a maintenance nightmare. All with a few clicks in a graphical user interface (GUI), you’ve turned a business challenge into an infrastructure benefit! *That’s* what an optimized infrastructure should look like from a management perspective—no projects, no hassles, just the ability to reconfigure IT assets smoothly, and with the click of a button. Does your infrastructure let you do that today?

Disaster Recovery Overhead

Having lots of database servers means having lots of databases, and having lots of databases means you have to find a way to back up and potentially restore all of that data. “Silo” data—that is, databases running on essentially standalone servers—is especially difficult to back up and restore simply because you’ve got so many management points. Each silo has its own storage, which means each server either needs dedicated backup equipment—even *more* resources you’ll need to manage—or you’ll burden the network with backup-related data. Some organizations create a separate network specifically to carry backup data to central backup servers—a network that involves even more resources that will need maintenance and management.

This is a classic problem of a fragmented, inflexible, *un-optimized* infrastructure. An optimized infrastructure, however, gives you centralized backup and restore capabilities for free, as part of being optimized.

Life Cycle Management

An un-optimized network plays havoc with hardware life cycle management. Older hardware is difficult to get rid of simply because it's so risky and time-consuming to move software such as SQL Server from one machine to another (in many cases, for example, client applications may have to be altered to use a new server name, or you might have to deal with significant downtime during a move or migration). Newer hardware is easy to add, but adds new overhead for management and maintenance.

To cope with these issues, organizations often try to have a standardized server platform, standardized storage platform, and so forth. The problem with this technique is that the hardware industry changes rapidly, and a “standardized” platform that is state-of-the-art today might not even be available for purchase in as little as a year.

Storage Management

Storage management has, for some time, been the bane of many organizations. Although SANs make storage capacity somewhat more dynamic, allocating that storage to SQL Server still requires relatively static designs. It's not always straightforward to expand or shrink the space made available to a given database, meaning you wind up designing each database with a “little extra” capacity—creating, in total, a *lot* of extra capacity—usually a lot more than you really need. Admittedly, storage management issues have improved vastly in the past few years, but when you start mixing in requirements for availability and the ways in which SQL Server needs everything to “mesh” together, storage management is still very much a challenge.

Performance Management

I've talked at length about the desire to engineer extra capacity into any SQL Server infrastructure; that's a valid and worthy goal, but simply adding more servers every time a new database requirement comes up isn't the answer. Doing so creates unacceptable bloat and “dedicated” extra capacity.

In other words, if Server A is running two databases at 40% capacity, the extra 60% is only available to those two databases. If Server B is running one database at 90% capacity, there is no way—using traditional architecture—to “transfer” the extra capacity from Server A to Server B. The result, then, is that *every* server must have extra capacity, creating a huge capacity surplus, albeit a surplus that is inflexible.

Here's an analogy: Imagine that every utility company you buy utilities from required payment in a different form of currency. You would have to maintain cash reserves in Euro, Dollars, Yen, and so forth—and imagine that there was no such thing as currency exchange. Many homeowners maintain a bit of a “pad” in their bank accounts for times of emergency—you'd be maintaining that “pad” in multiple different currencies, meaning your *total* “pad” would be several times what you might prefer. Wasted capacity, in other words.

Security and Auditing

And now the whopper: Securing and auditing your un-optimized infrastructure. Every server, every storage device, every network component, and every database—they all have to be secured, and organizations are increasingly subject to legal and industry requirements (to say nothing of internal policies) that require not only security but *verifiable, auditable* security. This is where you pay the price for all those “extra” resources in the infrastructure. Every single one is a major piece of overhead when it comes to securing and auditing.

In an optimized infrastructure, of course, it's easy. You simply tell the infrastructure what it should look like, from a security standpoint, and the various infrastructure elements dynamically comply. That is a large part of what Microsoft's Dynamic Systems Initiative (DSI) is all about—defining top-level policies and having managed elements automatically configure themselves to comply.

At a simpler level, however, you can ease security and auditing pains by simply reducing the number of resources on your network. *Consolidate*. Make every resource earn its place in the infrastructure by working to nearly maximum capacity so that every resource is absolutely necessary. That is another large part of what infrastructure optimization is all about.

Introducing the Infrastructure Optimization Model

Microsoft's Infrastructure Optimization Model (IOM) is a simple way for organizations to find out how optimized their infrastructure really is. The model uses common characteristics of IT organizations to classify the organization into one of four areas, shown in Figure 1.4: Basic, Standardized, Advanced, and Dynamic.

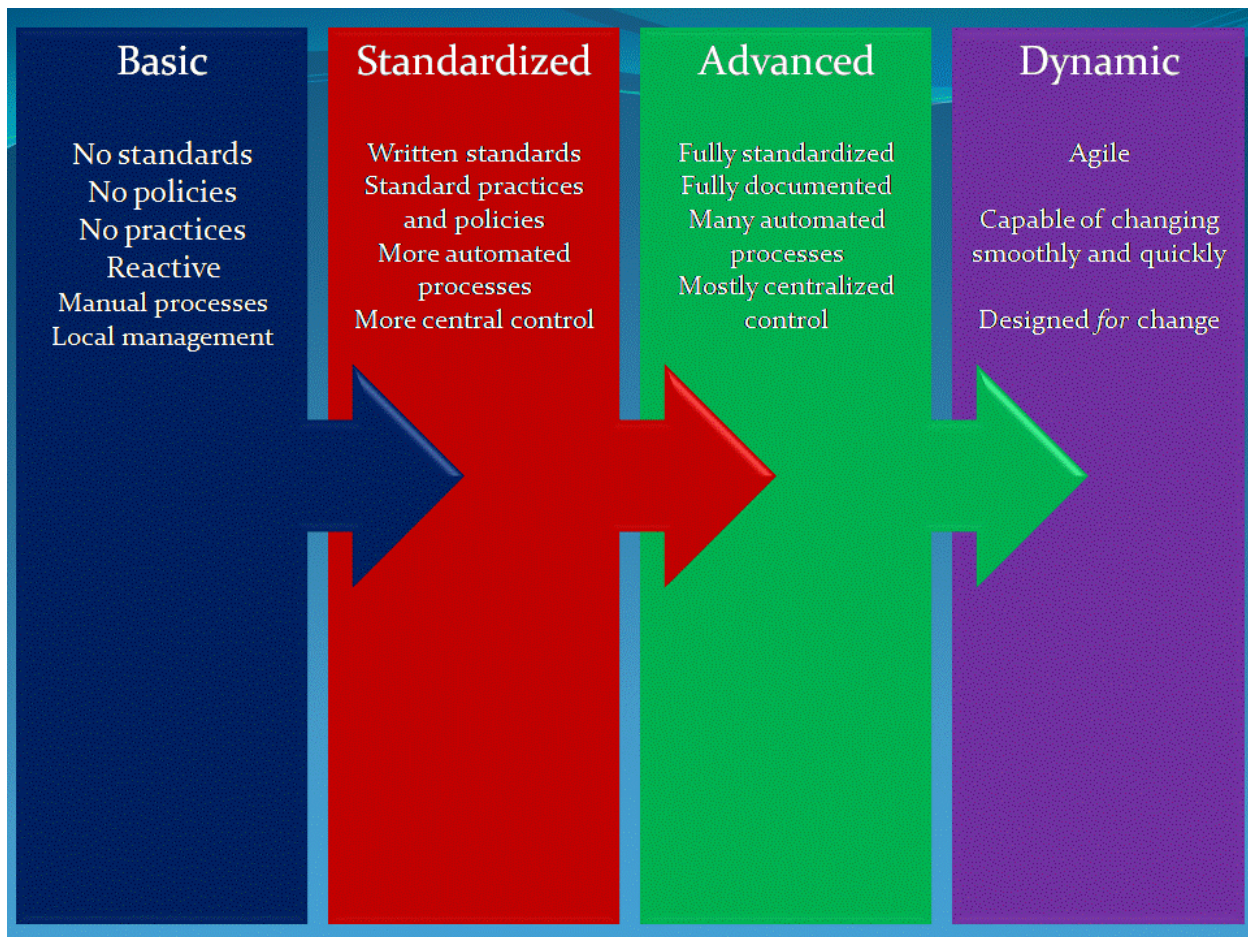


Figure 1.4: The Microsoft IOM.

Basic

An organization at the Basic level is essentially reactive, meaning they “fight fires.” The infrastructure is what it is, and most of the organization’s time is occupied dealing with problems and issues: little or no time may be available for new projects; the infrastructure contains wildly varying (non-standard) elements; and there may be no formal processes for change control, ensuring service levels, measuring service levels, and so forth. Processes tend to be manual and localized, with minimal central control over resources. Standards may not exist for backup, security, deployment, compliance, and other common practices.

Basic organizations tend to feature few tools, and the tools they do use are often nonstandard, assembled haphazardly over a long period of time. Desktop and server computers will run varying OSs, often at varying patch levels. Hardware is rarely standardized and legacy hardware and applications are commonly found. IT staff members may have little formal IT training or may simply be overwhelmed by the quantity of work. Very little central planning occurs, and new projects—if time is found for any—are often deployed haphazardly or over a long period of time. Many new deployments fail or spend forever in “planning” stages. IT is usually perceived as a cost center that significantly trails business requirements.

Standardized

An organization at the Standardized level is gaining control over its infrastructure. Written standards and policies have been introduced to govern common IT practices, to manage desktops, and so forth. Some centralization, such as for authentication, has taken place. An effort is being made to standardize resource types and configuration to make maintenance and management easier.

Fewer IT staffers are “jacks of all trades,” instead focusing on specific areas for administration. More specialization leads to more skilled staffers who make fewer mistakes and can focus more on automating common tasks. More tools are found in the Standardized organization, freeing up staffers to work on new projects and to implement new technologies and tools.

IT is still perceived as a cost center but it’s one that is well-managed. The Standardized organization may be able to allocate IT costs back to production business units, helping to justify their overhead; in most cases, business units will also have written service level agreements (SLAs) in place defining the quality of IT service they receive.

Advanced

An IT organization at the Advanced level is perceived as a business asset rather than as overhead. IT efforts are fully standardized, documented, and in most cases centrally controlled and managed. The cost of managing resources is at the lowest possible level due to standardization and well-written and executed IT policies. Security tends to be proactive, responding to threats and challenges rapidly and before they're realized in the form of production impact.

Management frameworks like the Information Technology Infrastructure Library (ITIL) are commonly a part of a Advanced organization. You can also expect to see management technologies, such as Microsoft's System Center family, providing centralized configuration and control. Deployment tools such as Windows Deployment Services (WDS) may also exist. In general, nearly every common IT task will be handled by some form of automated, centralized toolset—only exceptional tasks are handled manually in an Advanced organization.

Advanced IT organizations are seen as a business partner and an enabler for new business efforts. IT assets are still directly managed, although management is done through tools rather than locally and manually. Production business units turn to IT to help implement new business efforts and to reduce the cost of existing business efforts.

Dynamic

An organization at the Dynamic level exhibits the agility I've been mentioning throughout this chapter. The IT organization has implemented the tools, technologies, and techniques to quickly and smoothly reconfigure and redeploy IT resources on-demand, creating a flexible, malleable environment that can quickly respond to business needs—not only *reacting* to changing needs, but in fact *anticipating* those needs and enabling them to occur. Virtualization plays a large role, as does advanced forms of clustering, because they abstract software resources (such as applications) from the underlying hardware infrastructure, enabling software resources to be dynamically deployed and repositioned with little effort.

The tools used in a dynamic organization are more advanced. They tend to create a layer of abstraction between business policies and actual IT assets; deploying a new Active Directory (AD) domain controller, for example, may be as simple as selecting a machine and telling it to become a domain controller. The tools take care of reconfiguring the computer with not only the necessary software but also the necessary configuration standards for operations, security, auditing, and so forth. Changing a security setting for the organization may simply require a change to a policy; the tools take care of actually implementing that change on any computers that require it.

IT is perceived as a valuable partner in the business. IT creates the capability for new business efforts. It reduces overhead on existing business efforts, freeing up production business unit assets and management to focus on new directions; IT then implements additional capability and capacity to handle those new directions—all dynamically, all smoothly, and all nearly transparently. A major portion of IT staff time is spent on new projects because most routine maintenance is handled automatically and proactively. Little or no manual administration occurs.

Application Platform Optimization

Microsoft's Application Platform Optimization (APO) model is an application-specific look at the IOM, with a goal of providing the infrastructure, technologies, and tools needed to build more connected and adaptable systems. Within the APO model are several key areas, including:

- Data management—obviously the one most applicable to SQL Server
- Business intelligence
- Business process management
- Software development
- User experience

These five core capabilities ride “on top of” an optimized infrastructure, providing proactive capabilities that are corporate assets rather than cost centers.

Within the specific realm of data management, APO focuses on scalable, integrated database platforms designed to securely store and manage increasingly large quantities of data drawn from disparate sources—in other words, SQL Server. But, as I've discussed already, it's not enough to simply have SQL Server—it has to be running on an optimized infrastructure that is specifically designed to provide optimized infrastructure capabilities and characteristics to SQL Server.

Let's take roadways as an example. What does an “optimized roadway” look like? Well, that depends on what type of traffic it's going to carry. A major local thoroughfare, for example, might include a main highway for traffic that is passing through, with separate access roads leading to local homes and businesses. The access roads would periodically connect into the main highway, allowing traffic to interchange between “passing through” and “local access” lanes. That same level of optimization, however, wouldn't work for a major interstate, where the goal is to move larger amounts of traffic—including larger trucks—longer distances in a shorter amount of time. For a highway, specialized optimization technologies would be brought into play: Onramps and exits to firmly separate interstate traffic from local traffic; electronic metering systems to manage the merging of the two types of traffic at interchange points; bypasses for areas with closely spaced interchanges; and so forth. In other words, there is no single type of infrastructure that is suitable for all situations; the basic infrastructure model—the road—needs to be augmented with specific technologies designed to support the ways in which the infrastructure will be used. The same holds true for SQL Server, and in the next section, I'll lay out some of the specific keys that you'll need to look for in order to achieve a SQL Server-optimized infrastructure.

Keys for SQL Server Infrastructure Optimization

The next chapter will begin examining various ways of optimizing your infrastructure for SQL Server's benefit. I'll look at SQL Server's own technology and design parameters, technologies such as virtualization and clustering, techniques such as pooled-resource clusters, and so forth. Not all of these technologies and techniques are optimal for SQL Server, though, so I need to first define a set of criteria that we'll use to determine the appropriateness of any given technology or technique.

Abstracted

A key criterion is that SQL Server be abstracted from its underlying hardware. We need to manage SQL Server without regard for the underlying storage or other hardware resources. Ideally, we should be able to consider a large pool of servers, and a large pool of available storage, as a single unit. If we have the total server capacity to support 20,000 SQL Server connections (or physical input/output, or whatever other measurement you want to use), we should be able to allocate databases across that capacity at will. Similarly, if we have 10 terabytes of storage, we should be able to distribute that across databases as needed, with no regard for how that storage is physically implemented in the data center. Figure 1.5 illustrates this abstraction concept.

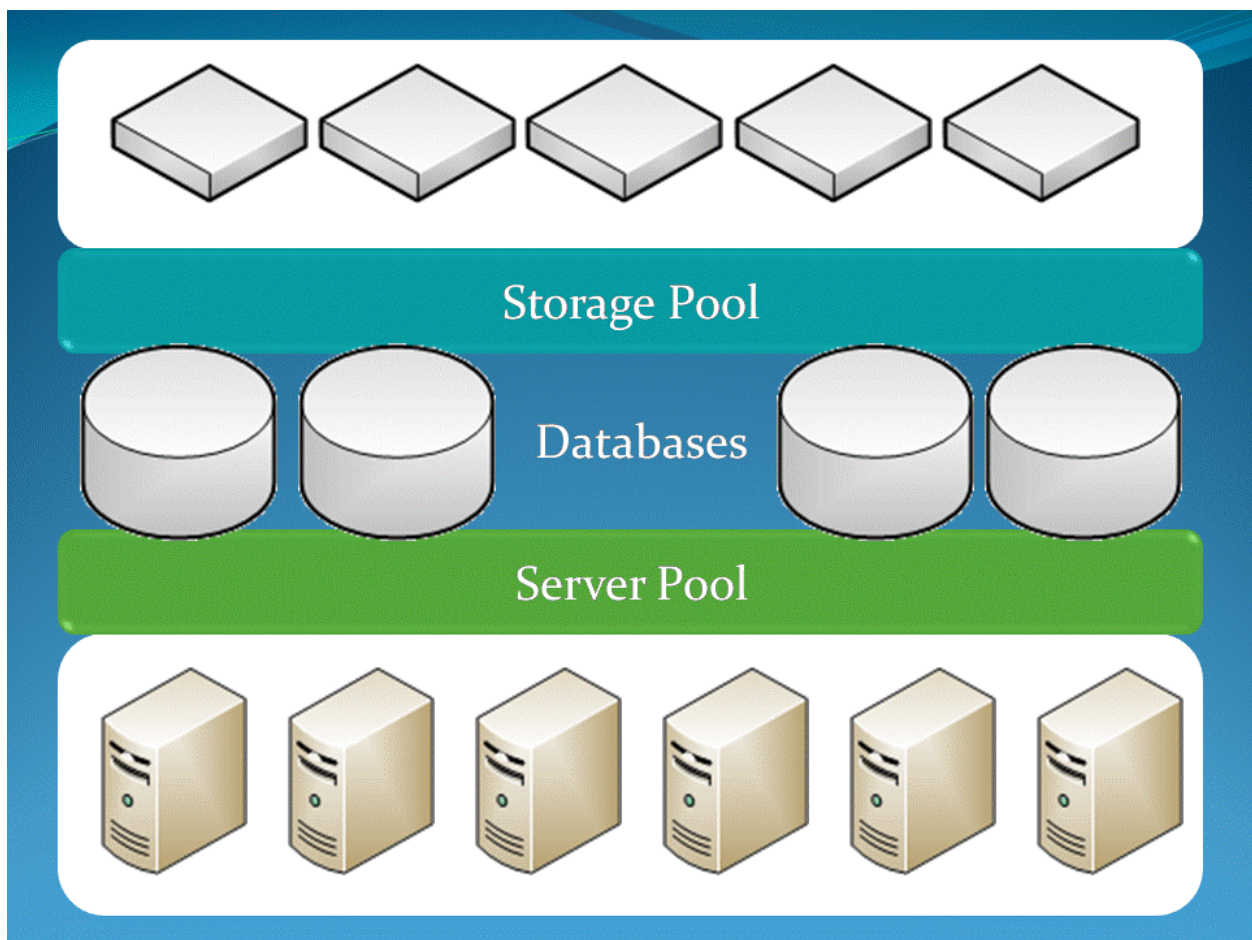


Figure 1.5: Abstracting SQL Server from the hardware resources.

Segmentation

We must be able to create multiple asset pools and assign databases to an entire pool. For example, we may have one pool for mission-critical applications, and that pool may have a larger percentage of extra capacity engineered into it (covered in following sections). Less-critical applications may be assigned to other asset pools with less extra capacity engineered in. Figure 1.6 illustrates the segmentation concept.

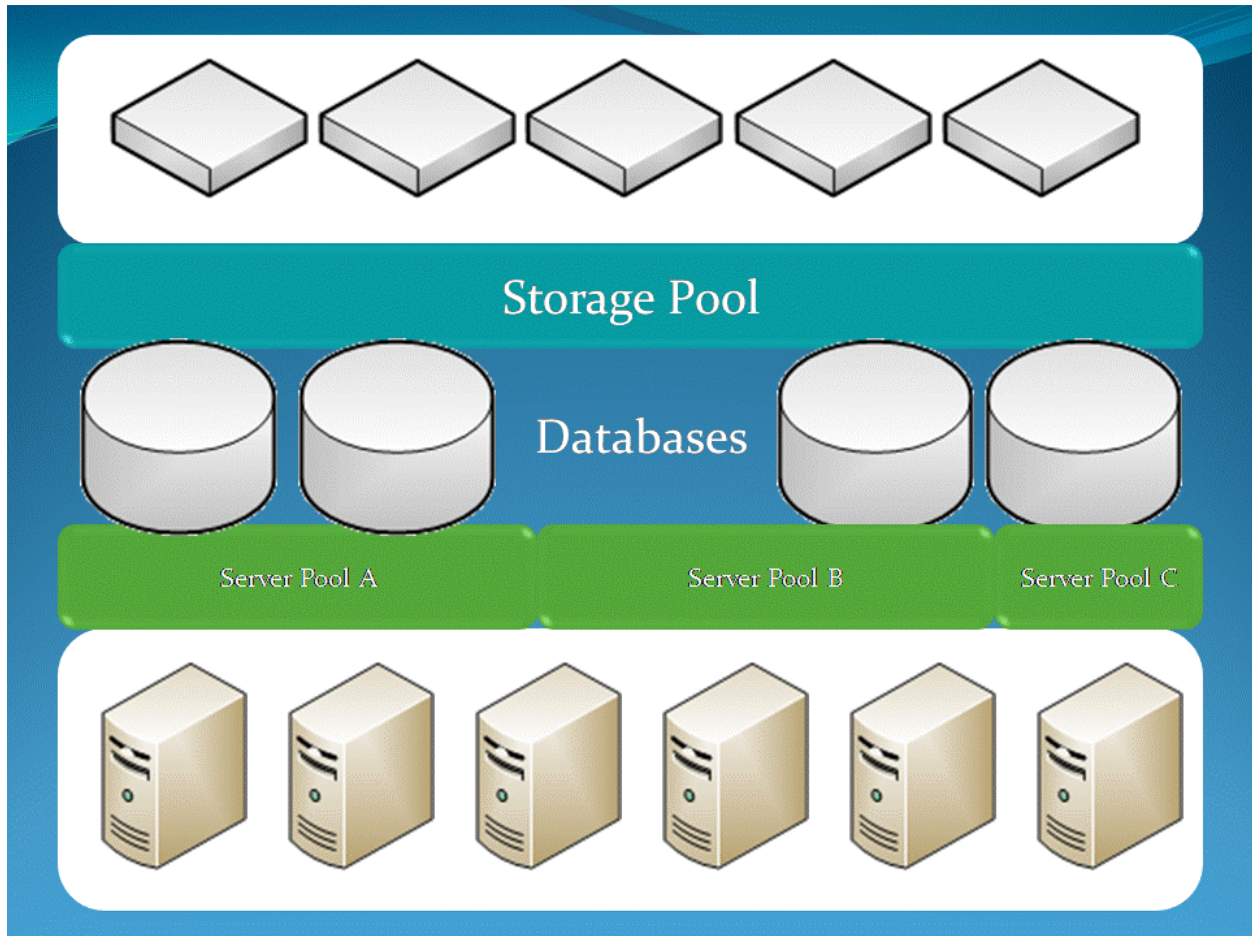


Figure 1.6: Segmenting assets into pools and assigning databases to pools.

Manageability

Management of SQL Server itself must be no more difficult than it is today. In other words, we can assume that any database will have a given amount of management overhead and that any single SQL Server computer (or, more accurately, SQL Server instance) will have a given amount of overhead. Whatever means are used to achieve our other goals should add no more management overhead than the existing database—and instance-related overhead we're already accustomed to. If possible, our solution should actually reduce the management overhead.

Ideally, management of SQL Server should become *easier*. An ideal solution would provide the capability to manage a “farm” of SQL Server computers as easily as one. Designating a single patch for installation, for example, would “push” the patch to all SQL Server computers as part of an overall process. Health monitoring would likewise be aggregated, and so forth, so that the entire collection of SQL Servers is “seen” as a single unit by administrators.

Performance

We must be able to achieve optimal SQL Server performance with little additional effort. We can rely on tools such as Microsoft System Center Operations Manager to tell us whether a given database or application is “healthy” or not; we must be able to correct “unhealthy” situations quickly and transparently—without taking the database or application offline.

This is a key requirement because it's one that most frequently leads to an un-optimized infrastructure using traditional SQL Server architectural techniques. It's easier to ensure good performance with an infinite number of servers, though, so a companion requirement is that we must have a *minimum of wasted capacity* in the new infrastructure. Any “margin for growth” capacity offered by the infrastructure must be offered in such a way that it is “growth capacity” for *all of our* databases, not just a particular one. In other words, we may design the infrastructure to have 10% extra capacity on purpose; that capacity must be immediately available to any database that needs it—the capacity cannot be “hard coded” to a specific database or SQL Server instance.

Availability

We must be able to maintain 100% availability of all databases at all times, through hardware failure, maintenance requirements, and so forth. We may, as a design decision, accept that a failure or maintenance action may reduce performance temporarily. Or we may engineer extra capacity into the infrastructure so that performance can be maintained to a specific level—understanding the requirement that any extra capacity must be universally available to all databases and SQL Server instances.

I fully acknowledge that most organizations do not offer 100% available for all of their databases in their SLAs. In some cases, then, this level of availability might be lower. I will suggest, however, that 100% available is what everyone *wants*—but they simply can't afford it using traditional techniques. So, for now, let's “aim high” for 100% availability all the time, and see what compromises we may need to make, or are willing to make, for lower levels of availability.

Life Cycle

We must be able to bring new server and storage assets into our infrastructure and remove old assets with no downtime and with minimal effort. For example, we should have the ability to deploy new servers with little manual setup beyond the Windows OS itself (which can also be automated, of course). Our pool of server assets should be able to be heterogeneous, permitting us to mix existing hardware assets and to bring in new assets based on market conditions, not on an arbitrary hardware standard.

Disaster Recovery

Our solution must provide robust support for disaster recovery, preferably using native SQL Server capability and/or industry-standard disaster recovery tools and techniques. Disaster recovery requirements should add an absolute minimum of additional assets to the infrastructure, and disaster recovery should work on conjunction with our pooled, abstracted hardware assets. We should have the capability of restoring to a purpose-built “recovery pool,” perhaps located off-site or hosted by a dedicated disaster recovery facility, and we should be able to independently engineer the performance, availability, and other aspects of this recovery pool. We should not *have* to give up our management advantages and dynamic capabilities (covered next) when restoring the infrastructure, even in the event of a total site failure.

Dynamic

Our infrastructure must be dynamic. We must be able to reallocate performance and storage assets transparently, with little or no impact to users or applications, on-demand. We must be able to dynamically shrink and expand server and storage pools, and dynamically reallocate databases and instances as desired. Reallocation should occur automatically in the event of hardware failure, but must be done so in a controllable and planned way. We must also be able to instantly reallocate any resource which was reallocated due to failure.

Coming Up...

With those criteria out of the way, let’s move on to the next chapter, where I’ll look at various ways of optimizing the infrastructure to support an application like SQL Server.