

Performance Guidance for SQL Server in Microsoft Azure Virtual Machines

SQL Server Technical Article

**Authors:** Silvano Coriani, Jasraj Dange, Ewan Fairweather, Xin Jin, Alexei Khalyako, Sanjay Mishra, Selcin Turkarslan

**Technical Reviewers:** Mark Russinovich, Brad Calder, Andrew Edwards, Suraj Puri, Flavio Muratore, Hanuma Kodavalla, Madhan Arumugam Ramakrishnan, Naveen Prakash, Robert Dorr, Roger Doherty, Steve Howard, Yorihito Tada, Kun Cheng, Chris Clayton, Igor Pagliai, Shep Sheppard, Tim Wieman, Greg Low, Juergen Thomas, Guy Bowerman, Evgeny Krivosheev

**Editor:** Beth Inghram

**Published:** June, 2013

**Updated:** September, 2014

**Applies to:** SQL Server and Microsoft Azure

**Summary:** Developers and IT professionals should be fully knowledgeable about how to optimize the performance of SQL Server workloads running in Microsoft Azure Infrastructure Services and in more traditional on-premises environments. This technical article discusses the key factors to consider when evaluating performance and planning a migration to SQL Server in Azure Virtual Machines. It also provides certain best practices and techniques for performance tuning and troubleshooting when using SQL Server in Microsoft Azure Infrastructure Services.

Copyright

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious.  No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2014 Microsoft. All rights reserved.

Contents

[Introduction 5](#_Toc398881139)

[Quick check list 5](#_Toc398881140)

[Azure Infrastructure Services fundamentals 6](#_Toc398881141)

[Azure VM configuration options 6](#_Toc398881142)

[Virtual machine size 6](#_Toc398881143)

[Network bandwidth 6](#_Toc398881144)

[Disk types and configurations 6](#_Toc398881145)

[Disk cache settings in Azure virtual machines 7](#_Toc398881146)

[Planning a virtual machine configuration in Azure for optimal SQL Server performance 9](#_Toc398881147)

[Different performance characteristics and considerations for major sub-systems between the cloud and on-premises 9](#_Toc398881148)

[I/O sub-system 9](#_Toc398881149)

[CPU and memory 10](#_Toc398881150)

[Network 10](#_Toc398881151)

[Raw storage performance testing 11](#_Toc398881152)

[Best practices and recommendations for optimizing SQL Server performance in Azure VMs 11](#_Toc398881153)

[Virtual machine sizes 11](#_Toc398881154)

[Azure virtual machine disks and cache settings 12](#_Toc398881155)

[Operating system disk vs. data disk 12](#_Toc398881156)

[Temporary disk 12](#_Toc398881157)

[Data disks performance options and considerations 12](#_Toc398881158)

[Single data disk configuration 13](#_Toc398881159)

[Multiple data disk configuration 13](#_Toc398881160)

[Adding multiple data disks to Azure virtual machine 13](#_Toc398881161)

[Disk striping options for Azure Virtual Machines 14](#_Toc398881162)

[Placement of database files 16](#_Toc398881163)

[TempDB 19](#_Toc398881164)

[Effects of warm-up on data disks 19](#_Toc398881165)

[Single vs. multiple storage accounts for data disks attached to a single VM 20](#_Toc398881166)

[NTFS allocation unit size 20](#_Toc398881167)

[Data compression for I/O bound workloads 20](#_Toc398881168)

[Restore performance – instant file initialization 22](#_Toc398881169)

[Other existing best practices 23](#_Toc398881170)

[Performance troubleshooting fundamental concepts 23](#_Toc398881171)

[Traditional factors that govern performance in SQL Server 24](#_Toc398881172)

[Factors that govern performance for SQL Server in Azure Virtual Machines 25](#_Toc398881173)

[Performance monitoring methodology for SQL Server in Azure Virtual Machine 26](#_Toc398881174)

[Appendix 30](#_Toc398881175)

[Raw storage performance testing scripts 30](#_Toc398881176)

[Guidance on defining key performance indicators 33](#_Toc398881177)

[How to use KPIs 33](#_Toc398881178)

[SQL Server troubleshooting scripts 35](#_Toc398881179)

[Snapshot wait stats script 35](#_Toc398881180)

[Requests executing on the system script 37](#_Toc398881181)

[Top query statements and plan by total CPU time 38](#_Toc398881182)

[Snapshot spinlock stats 39](#_Toc398881183)

[Snapshot I/O stats 41](#_Toc398881184)

[Performance monitor 44](#_Toc398881185)

# Introduction

The goal of this technical article is to provide guidance to developers and IT professionals on how to optimize the performance of Microsoft SQL Server workloads running in an Azure Virtual Machine environment. The article first describes the key concepts in Azure that have a direct impact on the performance of SQL Server workloads. It then introduces the key factors to take into account when evaluating performance and when you plan a migration to Azure platform. Next, it introduces additional considerations for performance troubleshooting in Azure Infrastructure Services. The article uses results from specific test scenarios to provide guidance and best practices for optimizing the performance of SQL Server workloads running in Azure virtual machine (VM).

The online documentation for SQL Server in Azure Virtual Machines covers getting started, migration, deployment, high availability, security, connectivity, backup and restore, creating a new virtual machine using one of our pre-built SQL Server platform images in the Image Gallery, and other topics. For more information, see [SQL Server in Azure Virtual Machines](http://msdn.microsoft.com/library/windowsazure/jj823132.aspx).

## Quick check list

The following is a quick check list that you can follow:

• Use minimum Standard Tier A2 for SQL Server VMs.

• Keep the storage account and SQL Server VM in the same region.

• Disable Azure geo-replication on the storage account.

• Avoid using operating system or temporary disks for database storage or logging.

• Avoid using Azure data disk caching options (caching policy = None).

• Stripe multiple Azure data disks to get increased IO throughput.

• Format with documented allocation sizes.

• Separate data and log file I/O paths to obtain dedicated IOPs for data and log.

• Enable database page compression.

• Enable instant file initialization for data files.

• Limit or disable autogrow on the database.

• Disable autoshrink on the database.

• Move all databases to data disks, including system databases.

• Move SQL Server error log and trace file directories to data disks.

• Apply SQL Server performance fixes.

• Setup default locations.

• Enable locked pages.

• Backup directly to blob storage.

For more information, please follow the guidelines provided in the following sub sections.

# Azure Infrastructure Services fundamentals

This section provides an overview of the Azure Infrastructure Services including some key considerations that can have a direct impact on the performance of your SQL Server workloads.

Azure Infrastructure Services lets you access scalable, on-demand infrastructure using Virtual Machines and Virtual Networks. You can use Azure to store data, to create virtual machines for development and test, and to build web applications or run other applications. Before you start reading this article, you should understand the fundamental concepts of Azure, such as cloud services, virtual machines, virtual networks, storage accounts, and so on. For more information, see [Azure Documentation](http://azure.microsoft.com/en-us/documentation/).

## Azure VM configuration options

### Virtual machine size

Azure virtual machines are available in different sizes and resource options in terms of number of CPU cores, memory capacity, maximum disk space, bandwidth, and so on. For the latest information about the supported virtual machine sizes and resource options, see [Virtual Machine and Cloud Service Sizes for Azure](http://msdn.microsoft.com/en-us/library/windowsazure/dn197896.aspx). We recommend that you review all these virtual machine size options and choose the ones best suited for your SQL Server workloads.

### Network bandwidth

The network traffic includes all traffic between client applications and SQL Server in Azure VM and, any other communication that involves a SQL Server process, or other processes in a virtual machine, such as ETL packages or backup and restore operations. We recommend that you choose a larger virtual machine size with more network bandwidth.

**Important note:** Network communications generated by accessing data disks and the operating system disk attached to the virtual machine are not considered part of the bandwidth limits.

### Disk types and configurations

Azure Virtual Machines provide three types of disks:

* **Operating system disk (persistent):** Every virtual machine has one attached operating system disk (**C:** drive) that has a limit of 127 GB. You can upload a virtual hard disk (VHD) that can be used as an operating system disk, or you can create a virtual machine from a platform image, in which case an operating system disk is created for you. An operating system disk contains a bootable operating system. It is mostly dedicated to operating system usage and its performance is optimized for operating system access patterns such as boot up.
* **Data disk (persistent)**: A data disk is a VHD that you can attach to a virtual machine to store application data. Currently, the largest single data disk is 1 terabyte (TB) in size. You can specify the disk size and add more data disks (up to 16, depending upon virtual machine size). This is useful when the expected database size exceeds the 1 TB limit or the throughput is higher than what one data disk can provide.
* **Temporary local disk (non-persistent):** Each virtual machine that you create has a temporary local disk, the **D:** drive and which is labeled as **TEMPORARY STORAGE**. This disk is used by applications and processes that run in the virtual machine for transient storage of data. It is also used to store page files for the operating system. Temporary disk volumes are hosted in the local disks on the physical machine that runs your virtual machine (VM). Temporary disk volumes are not persistent. In other words, any data on them may be lost if your virtual machine is restarted. Temporary disk volumes are shared across all other VMs running on the same host.

**Important note:** Proper configuration of your Azure disks (operating system and data) is one of the most important areas to focus on when optimizing the performance of your SQL Server workloads. Both operating system disks and data disks are stored as VHD files in the individual page blobs hosted in the Azure Storage Blob Service. Each blob has its own capacity limits based on the intrinsic storage architecture. For more information, see the following blog posts:

* [Azure Storage Scalability and Performance TargetsAzure](http://msdn.microsoft.com/en-us/library/azure/dn249410.aspx)
* [Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency](http://blogs.msdn.com/b/windowsazurestorage/archive/2011/11/20/windows-azure-storage-a-highly-available-cloud-storage-service-with-strong-consistency.aspx)
* [Data Series: Exploring Azure Drives, Disks, and Images](http://azure.microsoft.com/blog/2012/06/27/data-series-exploring-windows-azure-drives-disks-and-images/)
* [Erasure Coding in Azure Storage](http://blogs.msdn.com/b/windowsazurestorage/archive/2012/06/13/usenix-best-paper-award-erasure-coding-in-windows-azure-storage.aspx)

Each data disk can be a maximum of 1 TB in size. Depending upon your virtual machine size, you can attach up to 16 data disks to your virtual machine. I/Os per second (IOPS) and bandwidth are the most important factors to consider when determining how many data disks you need.

Storage capacity planning is similar to the traditional ‘spindle’ count in an on-premises storage design. In this article, we discuss various approaches you can use to spread I/O operations across one or more data disks, and discuss how to use SQL Server performance features, such as compression, to maximize the performance of your SQL Server workloads.

### Disk cache settings in Azure virtual machines

In Azure virtual machines, data of persisted drives is cached locally to the host machine, thus bringing the data closer to the virtual machine. Azure disks (operating system and data) use a two-tier cache:

* Frequently accessed data is stored in the memory (RAM) of the host machine.
* Less recently accessed data is stored on the local hard disks of the host machine. There is cache space reserved for each virtual machine operating system and data disks based on the virtual machine size.

Cache settings help reduce the number of transactions against Azure Storage and can reduce disk I/O latency in some scenarios.

Azure disks support three different cache settings: *Read Only*, *Read Write,* and *None (disabled)*. There is no configuration available to control the size of the cache.

* **Read Only:** Reads and writes are cached for future reads. All writes are persisted directly to Azure Storage to prevent data loss or data corruption while still enabling read cache.
* **Read Write:** Reads and writes are cached for future reads. Non-write-through writes are persisted to the local cache first, then lazily flushed to the Azure Blob service. For SQL Server, writes are always persisted to Azure Storage because it uses write-through. This cache setting offers the low disk latency for light workloads. It is recommended for the operating system and data disks with sporadic disk access. As the physical local disks are shared by all tenants on the host, you may observe increased latency when the total workload exceeds the performance of the physical host local disks.
* **None (disabled):** Requests bypass the cache completely. All disk transfers are completed against Azure Storage. This cache setting prevents the physical host local disks from becoming a bottleneck. It offers the highest I/O rate for I/O intensive workloads. Note that I/O operations to Azure Storage do incur transaction costs but I/O operations to the local cache do not.

The following table demonstrates the supported disk cache modes:

|  |  |  |  |
| --- | --- | --- | --- |
| **Disk type** | **Read Only** | **Read Write** | **None (disabled)** |
| Operating system disk | Supported | Default mode | Not supported |
| Data disk | Supported | Supported | Default mode |

Note that temporary disks are not included in this table because they are not Azure disks, but are implemented using local attached storage.

**Important notes:**

* Cache can be enabled for up to 4 data disks per virtual machine (VM).
* Changes to caching require a VM restart to take effect.

For specific recommendations and best practices for cache settings and performance behaviors, see the [Azure virtual machine disks and cache settings](#_Windows_Azure_virtual) section in this article.

# Planning a virtual machine configuration in Azure for optimal SQL Server performance

Before you migrate your SQL Server workloads to Azure Infrastructure Services, consider the hardware, design, and architectural differences between your on-premises environment and Azure Infrastructure Services. Knowing these differences will help you identify which workloads are appropriate for this environment. The [Microsoft Assessment and Planning (MAP) Toolkit](http://microsoft.com/map) can help you assess your current IT infrastructure for various platform migrations, including Azure Virtual Machines. You should also consider adopting an approach that is well suited to measure the performance of your SQL Server workloads in Azure Infrastructure Services.

## Different performance characteristics and considerations for major sub-systems between the cloud and on-premises

### I/O sub-system

Storage optimization for SQL Server transactional and analytical workloads is a very important task that requires careful planning and analysis. There is already a tremendous amount of information that explains how to deal with I/O subsystems with different performance characteristics in a traditional on-premises environment, such as spindles, host bus adapters (HBAs), disk controllers, and so on. For more information, see [Analyzing I/O Characteristics and Sizing Storage Systems for SQL Server Database Applications](http://msdn.microsoft.com/en-us/library/ee410782.aspx).

Azure disks are implemented as a service, so they do not offer the same range of complex configuration options available in traditional on-premises I/O subsystems. This has both benefits and costs. For instance, Azure disks offer built-in local redundancy and optional geo-redundancy for disaster recovery through the use of replicas. To achieve the same level of redundancy in on-premises deployments, you would need to set up multiple disk arrays in multiple locations and a synchronization mechanism, such as, a storage area network (SAN) replication. On the other hand, the Azure disk performance is not as predictable as on-premises disk I/O subsystem due to several factors:

* Azure Infrastructure Services is a shared, multi-tenant service. Resources like host machines, storage services and network bandwidth are shared among multiple subscribers.
* Performance may vary depending upon where and when you provision your virtual machines due to a variety of factors including differences in hardware. Your virtual machine may get moved to a different host due to a hardware replacement necessitated by a failure or lifecycle refresh.
* The performance and availability of your virtual machines may be impacted (positively or negatively) by maintenance operations such as platform upgrades or performance and reliability fixes.
* When you use cloud-based storage options in Azure, you sacrifice granular control and deep performance optimization options for lower costs, simplicity and out-of-the-box redundancy.
* Azure disks are connected to virtual machines via a network infrastructure and that can introduce higher network latency compared to the local attached disks in on-premises environment.

For a detailed discussion on different storage configurations, see the [Best practices and recommendations for optimizing SQL Server performance in Azure VMs](#_Best_practices_and) section.

### CPU and memory

Although Azure Storage introduces most of the differences between on-premises and Azure deployments, other system components, such as CPU and memory, need to be considered as well when you evaluate performance. In Azure, the only configurable option for CPU is the number of CPU cores assigned to a virtual machine deployment, currently from one shared (A0) to sixteen dedicated in A9 in the Standard Tier, which is subject to increase or change in the future. These CPU cores may not be the same as the SKUs customers can find in expensive on-premises servers. This can lead to significant performance differences in CPU-bound workload and needs to be taken into account during testing and baselining.

For memory, the current offering spans from 768 MB in A0 to 112GB in an A9 Compute Intensive Instance in the Standard Tier. Again, when you compare the performance characteristics of an existing on-premises application, you need to consider the right virtual machine size and its other options to avoid any performance impact due to inadequate memory sizing for buffer pools or other internal memory structures.

### Network

Network latency in Azure Infrastructure Services can be higher than that of a traditional on-premises environment, due to several reasons, such as virtualization, security, load balancers, and so on. This means that reducing network round trips between application layers in a cloud solution has a strong positive impact on the performance when compared to on-premises solutions.

For “chatty” applications, where communications between application layers and components are frequent, we recommend that you consolidate multiple application layers on the same virtual machine. This reduces the number of tiers and the amount of communications that your application needs resulting in better performance.

When this is not possible or even counterproductive, consider the following recommendations to optimize network communication:

* Host virtual machines in the same cloud service to enable direct communication between virtual machines via internal IP addresses. For more information, see [Tutorial: Connect to SQL Server in the same cloud service](http://msdn.microsoft.com/en-us/library/windowsazure/jj823136.aspx).
* Use Azure Virtual Network for virtual machines that reside in different cloud services. For more information, see [Virtual Networks](https://www.windowsazure.com/en-us/manage/services/networking/) and [Tutorial: Connect ASP.NET application to SQL Server in Azure via Virtual Network](http://msdn.microsoft.com/en-us/library/windowsazure/jj823135.aspx).
* Load balance multiple virtual machines in the same cloud service via public virtual IP addresses. For more information, see [Load Balancing Virtual Machines](https://www.windowsazure.com/en-us/manage/windows/common-tasks/how-to-load-balance-virtual-machines/).

## Raw storage performance testing

To achieve optimal results in any complex storage environment, you need to follow a systematic, data- driven methodology. This section summarizes the approach that we use internally within Microsoft to develop best practices on how to configure Azure Infrastructure Services and SQL Server for various types of workloads.

To perform I/O benchmarking tests on different Azure disk configurations, we use the Microsoft [SQLIO Disk Subsystem Benchmark Tool](http://www.microsoft.com/en-us/download/details.aspx?id=20163) and analyze the various performance metrics as described in the [Analyzing I/O Characteristics and Sizing Storage Systems for SQL Server Database Applications](http://msdn.microsoft.com/en-us/library/ee410782.aspx) article through the various performance counters and dynamic management views (DMVs). The primary goal of these tests is to determine the I/O capacity of various configurations. Then, we compare these configurations with different SQL Server workloads (for example, OLTP, point lookup queries, analytical queries with aggregations) and I/O patterns. Our goal is to verify which configuration is most appropriate for each type of workload. The [Best practices and recommendations for optimizing SQL Server performance in Azure VMs](#_Best_practices_and) section contains our findings and recommendations based on these results.

Note that precise I/O performance depends on a number of factors including data center capacity and network utilization. So, you should consider these generalized findings as subject to change. We hope that they will be useful for capacity planning as you consider which SQL Server workloads you want to migrate to Azure environment.

We recommend that you also run your own validation tests. You can see a detailed example of reusable scripts in the [Appendix](#_Appendix).

# Best practices and recommendations for optimizing SQL Server performance in Azure VMs

Many of the same techniques used to optimize SQL Server performance in your on-premises environment can be used to tune your SQL Server workloads in Azure Infrastructure Services. Having said that, running your database workload in a hosted multi-tenant cloud service like Azure Infrastructure Services is fundamentally different and if you want to be successful you will need to consider some new best practices. This section provides new best practices and recommendations for optimizing SQL Server performance in Azure Infrastructure Services.

## Virtual machine sizes

In general, smaller VM sizes are best suited for lightweight development and test systems or applications. For production and intensive workload deployments, bigger virtual machine sizes (such as A3 or high memory instances) are often a better choice because they can provide more capacity in terms of virtual cores, memory and data disks.

For SQL Server production workloads, we recommend that you use minimum Standard Tier A2 VM sizes or bigger instances. Starting with May 2014, new VM sizes (A8 and A9) have been introduced sporting faster Intel Xeon processor and increased memory sizes. Based on various performance tests, these VMs provide important benefits in terms of CPU performance, IO throughput and bandwidth. If you plan to run very high SQL Server workloads in Azure Virtual Machines, we recommend that you consider these new VM sizes.

## Azure virtual machine disks and cache settings

Azure Virtual Machines provide three types of disks: operating system (OS) disk, temporary disk, and data disks. For a description of each disk type, see section [Azure Infrastructure services fundamentals](#_Windows_Azure_Infrastructure) in this article.

### Operating system disk vs. data disk

When placing your data and log files you should consider disk cache settings in addition to size limits. For a description of cache settings, see section [Azure Infrastructure services fundamentals](#_Windows_Azure_Infrastructure) in this article.

While “Read Write” cache (default setting) for the operating system disk helps improve the overall operating system performance, boot times and reducing the read latency for the IO patterns that OS usually generates, we recommend that you do not use OS disk for hosting system and user database files. Instead, we recommend that you use data disks. When the workload demands a high rate of random I/Os (such as a SQL Server OLTP workload) and throughput is important to you, the general guideline is to keep the cache set to the default value of “None” (disabled). Because Azure storage is capable of more IOPS than a direct attached storage disk, this setting causes the physical host local disks to be bypassed, therefore providing the highest I/O rate.

### Temporary disk

Unlike Azure disks (operating system and data disks) which are essentially VHDs stored as page blobs in Azure Storage, the temporary disk (labeled as **D:**) is not persistent and is not implemented using Azure Storage. It is reserved by the operating system for the page file and its performance is not guaranteed to be predictable. Any data stored on it may be lost after your virtual machine is restarted or resized. Hence, we do not recommend the **D:** drive for storing any user or system database files, including **tempdb**.

## Data disks performance options and considerations

This section discusses the best practices and recommendations on data disk performance options based on testing done by Microsoft. You should be familiar with how SQL Server I/O operations work in order to interpret the test results reported in this section. For more information, see [Pages and Extents Architecture](http://msdn.microsoft.com/en-us/library/cc280360(v=sql.105).aspx).

It is important to note that the results we provide in this section were achieved without [SQL Server High Availability and Disaster Recovery Solutions](http://msdn.microsoft.com/library/windowsazure/jj870962.aspx) enabled (such as, AlwaysOn Availability Groups, database mirroring or log shipping). We recommend that you deploy one of these features to maintain multiple redundant copies of your databases across at least two virtual machines in an availability set in order to be covered by the [Azure Cloud Services, Virtual Machines, and Virtual Network Service Level Agreement](http://www.microsoft.com/en-us/download/details.aspx?id=38427). Enabling any of these features affects performance, so you should consider incorporating one of them in your own performance testing to get more accurate results.

As a general rule, we recommend that you attach maximum number of disks allowed by the VM size (such as, 16 data disks for an A7 VM) for throughput sensitive applications. While latency may not necessarily improve by adding more data disks when your workload is within the maximum IOPS limit, the additional IOPS and bandwidth that you get from the attached additional disks can help to avoid reaching the single disk 500 IOPS limit. Note that this might trigger throttling events that might increase disk response times and disk latency.

### Single data disk configuration

In our performance tests, we’ve executed several SQL Server I/O measurements to understand data disk response characteristics with respect to the typical I/O patterns generated by SQL Server based on different kind of workloads. The results for a single disk configuration on an A7 VM instance are summarized here:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Random I/O  (8 KB Pages) | | | | Sequential I/O  (64 KB Extents) | |
|  | | **Reads** | **Writes** | **Reads** | | **Writes** |
| IOPS | | 500 | 500 | 500 | | 300 |
| Bandwidth | | 4 MB/s | 4 MB/s | 30 MB/s | | 20 MB/s |

**Note:** Because Azure Infrastructure Services is a multi-tenant environment, performance results may vary. You should consider these results as an indication of what you can achieve, but not a guarantee. We suggest you repeat these tests and measurements based on your specific workload.

### Multiple data disk configuration

If your workload exceeds or is close to the I/O performance numbers mentioned in the previous section, we recommend that you add multiple disks (depending on your virtual machine size) and stripe multiple disks in volumes. This configuration gives you the ability to create volumes with specific throughput and bandwidth, based on your data and log performance needs by combining multiple data disks together.

### Adding multiple data disks to Azure virtual machine

After you create a virtual machine in Azure, you can attach a data disk to it using either the Azure Management Portal or the [Add-AzureDataDisk](http://msdn.microsoft.com/en-us/library/windowsazure/jj152837.aspx) Azure PowerShell cmdlet. Both techniques allow you to select an existing data disk from a storage account, or create a new blank data disk.

If you choose to create a new blank data disk at the Management Portal, you can choose the storage account that your virtual machine was created in but not a different storage account.

To place your existing data disk (.vhd file) into a specific storage account, you need to use the Azure PowerShell cmdlets. The following example demonstrates how to update a virtual machine using the [Get-AzureVM](http://msdn.microsoft.com/library/dn495236.aspx) and the [Add-AzureDataDisk](http://msdn.microsoft.com/library/dn495298.aspx) cmdlets. The [Get-AzureVM](http://msdn.microsoft.com/en-us/library/windowsazure/jj152849.aspx) cmdlet retrieves information on a specific virtual machine. The Add-AzureDataDisk cmdlet creates a new data disk with specified size and label in a previously created Storage Account.

Get-AzureVM "CloudServiceName" -Name "VMNAme" | Add-AzureDataDisk -CreateNew -DiskSizeInGB 100 -MediaLocation ` "https://<storageaccount>.blob.core.windows.net/vmdisk/Disk1.vhd" -DiskLabel "disk1" -LUN 1 | Update-AzureVM

To create a new storage account, use the [New-AzureStorageAccount](http://msdn.microsoft.com/en-us/library/dn495115.aspx) cmdlet as follows:

New-AzureStorageAccount -StorageAccountName "StorageAccountX" -Label "StorageAccountX" -Location "North Central US"

For more information about Azure PowerShell cmdlets, see [Azure PowerShell](http://msdn.microsoft.com/en-us/library/windowsazure/jj156055.aspx) on MSDN and [Azure command line tools](https://www.windowsazure.com/en-us/downloads/).

### Disk striping options for Azure Virtual Machines

For Azure VMs running on Windows Server 2008 R2 and previous releases, the only striping technology available is [striped volumes for dynamic disks](http://technet.microsoft.com/en-us/library/cc732422.aspx). You can use this option to stripe multiple data disks into volumes that provide more throughput and bandwidth than what a single disk can provide.

Starting with Windows Server 2012, [Storage Pools](http://technet.microsoft.com/library/hh831739.aspx) are introduced and operating system software RAID capabilities are deprecated. Storage Pools enable you to virtualize storage by grouping industry-standard disks into “pools”, and then create virtual disks called *Storage Spaces* from the available capacity in the storage pools. You can then configure these virtual disks to provide striping capabilities across all disks in the pool, combining good performance characteristics. In addition, it enables you to add and remove disk space based on your needs.

During our tests, after adding a number of data disks (4, 8 and 16) as shown in the previous section, we created a new storage pool by using the following Windows PowerShell command:

New-StoragePool –FriendlyName StoragePool1 –StorageSubsystemFriendlyName "Storage Spaces\*" –PhysicalDisks (Get-PhysicalDisk –CanPool $True)

Next, we created a virtual disk on top of the new storage pool and specified resiliency setting and virtual disk size.

$disks = Get-StoragePool –FriendlyName StoragePool1 -IsPrimordial $false | Get-PhysicalDisk

New-VirtualDisk –FriendlyName VirtualDisk1 -ResiliencySettingName Simple –NumberOfColumns $disks.Count –UseMaximumSize –Interleave 256KB

Important Note:For performance, it is very important that the –**NumberOfColumns** parameter is set to the number of disks utilized to create the underlying Storage Pool. Otherwise, IO requests cannot be evenly distributed across all data disks in the pool and you will get suboptimal performance.

The **–Interleave** parameter enables you to specify the number of bytes written in each underlying data disk in a virtual disk. We recommend that you use 256 KB for all workloads.

Lastly, we created and formatted the volume to make it usable to the operating system and applications by using the following Windows PowerShell commands:

Get-VirtualDisk –FriendlyName VirtualDisk1 | Get-Disk | Initialize-Disk –Passthru | New-Partition –AssignDriveLetter –UseMaximumSize | Format-Volume –AllocationUnitSize 64K

Once the volume created, it is possible to dynamically increase the disk capacity by attaching new data disks. To achieve optimal capacity utilization, consider the number of columns your storage spaces have and add disks in multiples of that number. See [Windows Server Storage spaces Frequently Asked Questions](http://social.technet.microsoft.com/wiki/contents/articles/11382.storage-spaces-frequently-asked-questions-faq.aspx) for more information.

Using Storage Pools instead of traditional Windows operating system striping in dynamic disks brings several advantages in terms of performance and manageability. We recommend that you use Storage Pools for disk striping in Azure Virtual Machines.

During our internal testing, we have implemented the following scenarios with different number of disks as well as disk volume configurations. We tested the following scenarios with configurations of 4, 8 and 16 data disks respectively, and we observed increased IOPS for each data disk added as expected:

* We arranged multiple data disks as simple volumes and leveraged the [Database Files and Filegroups](http://msdn.microsoft.com/library/ms189563.aspx) feature of SQL Server to stripe database files across multiple volumes.
* We used Windows Server Storage Pools to create larger volumes, which contains multiple data disks, and we placed database and log files inside these volumes.

It’s important to notice that using multiple data disks provides performance benefits but it creates more management overhead. In addition, partial unavailability of one of the striped disks can result in unavailability of a database. Therefore, for such configurations, we recommend that you consider enhancing the availability of your databases using high availability and disaster recovery capabilities of SQL Server as described in [High Availability and Disaster Recovery for SQL Server in Azure Virtual Machines](http://msdn.microsoft.com/library/windowsazure/jj870962.aspx).

The following tables summarize the results of tests that we performed using multiple data disks configurations at Microsoft.

**Aggregated throughput and bandwidth across 4 data disks**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Random I/O  (8 KB Pages) | | | | Sequential I/O  (64 KB Extents) | |
|  | | **Reads** | **Writes** | **Reads** | | **Writes** |
| IOPS | | 2000 | 2000 | 1600 | | 1200 |
| Bandwidth | | 16 MB/s | 16 MB/s | 100 MB/s | | 75 MB/s |

**Aggregated throughput and bandwidth across 8 data disks**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Random I/O  (8 KB Pages) | | | | Sequential I/O  (64 KB Extents) | |
|  | | **Reads** | **Writes** | **Reads** | | **Writes** |
| IOPS | | 4000 | 4000 | 2400 | | 2400 |
| Bandwidth | | 30 MB/s | 30 MB/s | 150 MB/s | | 150 MB/s |

**Aggregated throughput and bandwidth across 16 data disks**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Random I/O  (8 KB Pages) | | | | Sequential I/O  (64 KB Extents) | |
|  | | **Reads** | **Writes** | **Reads** | | **Writes** |
| IOPS | | 8000 | 8000 | 2400 | | 4000 |
| Bandwidth | | 60 MB/s | 60 MB/s | 150 MB/s | | 250 MB/s |

**Note:** Because Azure Infrastructure Services is a shared, multi-tenant environment, performance results may vary. You should consider these results as an indication of what you can achieve, but not a guarantee. We recommend that you repeat these tests and measurements based on your specific workload.

By using the newly introduced Intel-based A8 and A9 VM sizes, we repeated our IO performance tests and noticed a significant increase in bandwidth and throughput for larger sequential IO requests. If you use Intel-based A8 and A9 VM sizes, you can get a performance increase for 64 KB (and bigger) read and write operations. If your workload is IO intensive, these new VM sizes (A8 and A9) can help in achieving more linear scalability compare to smaller VM sizes, but always within the 500 IOPs per disk boundaries. For more information, see [About the A8 and A9 Compute Intensive Instances](http://msdn.microsoft.com/library/azure/dn689095.aspx).

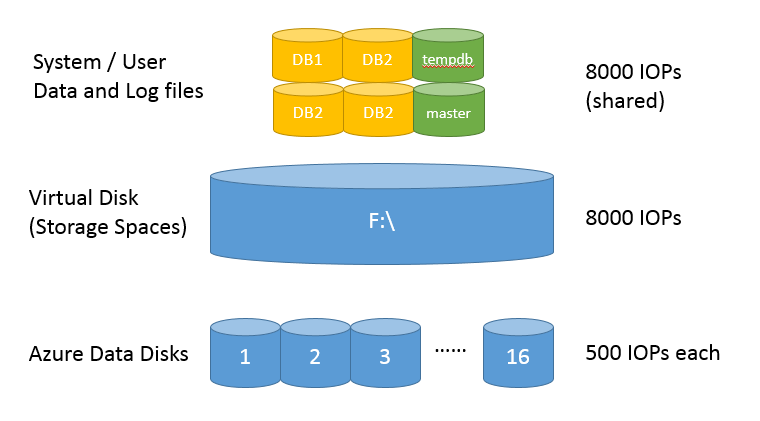
Based on our tests, we have made the following observations about the Azure Virtual Machine environment:

* Spreading your I/O workload across a number of data disks benefits smaller random operations (more common in OLTP scenarios) where IOPS and bandwidth scale in a nearly linear fashion.
* As the I/O block size increases, for read operations adding more data disks does not result in higher IOPS or bandwidth. This means that if your workload is read intensive with more analytical queries, adding more disks will not necessarily help.
* For write intensive workload, adding more data disks can increase performance in a nearly linear fashion. This means that you can benefit from placing each transaction log for multiple databases on a separate data disk.
* For large sequential I/O block sizes (such as, 64 KB or greater), writes generally perform better than reads.
* A8 and A9 VM sizes provide increased throughput for IO sensitive workloads.

### Placement of database files

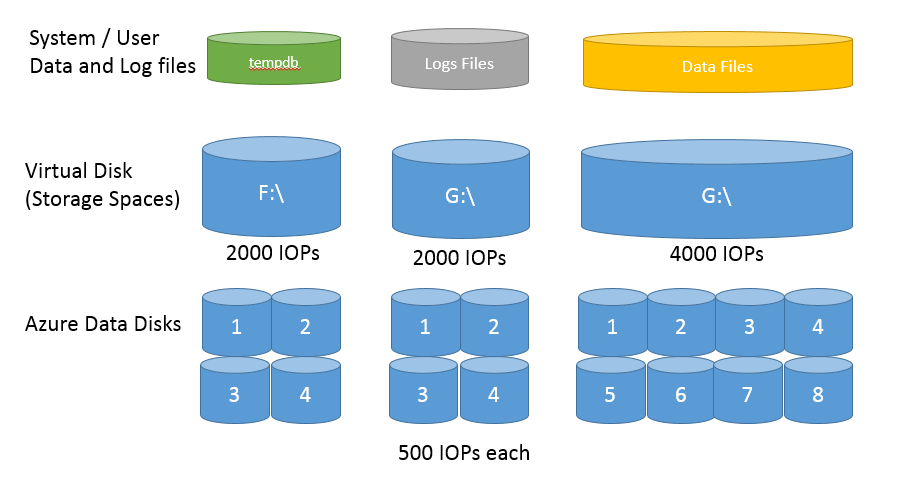
Depending on how you configure your storage, you should place and the data and log files for user and system databases accordingly to achieve your performance goals. This section provides guidance on how you should place database files when using SQL Server in Azure Virtual Machines:

* **Option 1:** You can create a **single striped volume** using Windows Server Storage Spaces leveraging multiple data disks, **and place all database and log files in this volume**. In this scenario, all your database workload shares aggregated I/O throughput and bandwidth provided by these multiple disks, and you simplify the placement of database files. Individual database workloads are load balanced across all available disks, and you do not need to worry about single database spikes or workload distribution. You can find the graphical representation of this configuration below:



* **Option 2:** You can create **multiple striped volumes**, each composed by the number of data disks required to achieve specific I/O performance, **and do a careful placement of user and system database files on these volumes** accordingly. You may have one important production database with a write-intensive workload that has high priority, and you may want to maximize the database and log file throughput by segregating them on two separate 4 disk volumes (each volume providing around 2000 IOPs and 100 MB/sec). For example, use:
* 4-disks volume for hosting TempDB data and log files.
* 4-disks volume for hosting other minor databases.

This option can give you precise file placement by optimizing available IO performance. You can find the graphical representation of this configuration below:



You can still create single disk volumes and leverage SQL Server files and filegroups placement for your databases. While this can still offer some benefits in terms of flexible storage layout organization, it introduces additional complexity and also limits single file (data or log) IO performance to a value that a single Azure data disk can provide such as 500 IOPs and 60 MB/sec.

Although Azure data disks have different behaviors than traditional rotating spindles (,in which competing random and sequential operations on the same disks can impact performance), we still recommend that you keep data and log files in different paths to achieve dedicated IOPs and bandwidth for them.

To help understand your IO requirements and performance while running your SQL Server workloads on Azure Virtual Machines, you need to analyze the following three tools and combine the results carefully:

* [**SQL Server IO statistics**](#_Snapshot_I/O_stats)**:** They reflect the database management system view of the IO subsystem.
* [**Windows Server Logical Disk Performance Counters**](#_Performance_monitor): They show how the operating system performs on IOs.
* [**Azure Storage Analytics**](http://msdn.microsoft.com/en-us/library/azure/hh343270.aspx)**:** Azure hosts data disks’ VHD files in Azure Storage. You can turn on logging and metrics for the storage account that hosts your data disks, and get useful information such as the number of successful and failed requests, timeout, throttling, network, authorization, and other errors. You can configure and get data from these metrics on the [Azure Portal](http://azure.microsoft.com/en-us/documentation/articles/storage-monitor-storage-account/), or via PowerShell, REST APIs, and .NET Storage Client library.

By leveraging all these information, you can understand:

* If your IO related stalls or wait types in SQL Server (manifesting as increased disk response times in OS Perf Counters) are related to throttling events happening in Azure Storage. And,
* If rebalancing your data and log files across different volumes (and underlying disks) can help maintaining throughput and bandwidth between storage performance limits.

### TempDB

As mentioned in section [Azure virtual machine disks and cache settings](#_Windows_Azure_virtual), we recommend that you place **tempDB** on data disks instead of the temporary disk (D:). Following are the three primary reasons for this recommendation based on our internal testing with SQL Server test workloads.

* **Performance variance:** In our testing, we noticed that you can get the same level of performance you get on D:, if not more IOPS, from a single data disk. However, the performance of D: drive is not guaranteed to be as predictable as the operating system or data disk. This is because the size of the D: drive and the performance you get from it depends on the size of the virtual machine you use, and the underlying physical disks shared between all VMs hosted by the same server.
* **Configuration upon VM downtime situation:** If the virtual machine gets shutdown down (due to planned or unplanned reasons), in order for SQL Server to recreate the tempDB under the D: drive, the service account under which SQL Server service is started needs to have local administrator privileges. In addition, the common practice with on-premises SQL deployments is to keep database and log files (including tempDB) in a separate folder, in which case the folder needs to be created before SQL Server starts. For most customers, this extra re-configuration overhead is not worth the return.
* **Performance bottleneck:** If you place tempdb on D: drive and your application workloads use tempDB heavily, this can cause performance bottleneck because the D: drive can introduce constraints in terms of IOPS throughput. Instead, place tempDB on data disks to gain more flexibility. For more information on configuration best practices for optimizing tempdb, see [Compilation of SQL Server TempDB IO Best Practices](http://blogs.msdn.com/b/cindygross/archive/2009/11/20/compilation-of-sql-server-tempdb-io-best-practices.aspx).

We strongly recommend that you perform your own workload testing before implementing a desired SQL Server file layout strategy.

### Effects of warm-up on data disks

With Azure disks, we have observed a “warm-up effect” that can result in a reduced rate of throughput and bandwidth for a short period of time. In situations where a data disk is not accessed for a period of time (approximately 20 minutes), adaptive partitioning and load balancing mechanisms kick in. If the disk is accessed while these algorithms are active, you may notice some degradation in throughput and bandwidth for a short period of time (approximately 10 minutes), after which they return to their normal levels. This warm-up effect happens because of the adaptive partitioning and load balancing mechanism of Azure, which dynamically adjusts to workload changes in a multi-tenant storage environment. You may observe similar effects in other widely known cloud storage systems as well. For more information, see [Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency](http://blogs.msdn.com/b/windowsazurestorage/archive/2011/11/20/windows-azure-storage-a-highly-available-cloud-storage-service-with-strong-consistency.aspx).

This warm-up effect is unlikely to be noticed for systems that are in continuous use. But we recommend you consider it during performance testing or when accessing systems that have been inactive for a while.

### Single vs. multiple storage accounts for data disks attached to a single VM

To simplify management and reduce potential risks of consistency in case of failures, we recommend that you leave all the data disks attached to a single virtual machine in the same storage account. Storage accounts are implemented as a recovery unit in case of failures. So, keeping all the disks in the same account makes the recovery operations simple. There is no performance improvement if you store data disks attached to a single VM in multiple storage accounts. If you have multiple VMs, we recommend that you consider the storage account limits for throughput and bandwidth during capacity planning. In addition, distribute VMs and their data disks to multiple storage accounts if the aggregated throughput or bandwidth is higher than what a single storage account can provide. For information on storage account limits, see [Azure Storage Scalability and Performance Targets](http://msdn.microsoft.com/library/dn249410.aspx). For information on max IOPS per disk, see [Virtual Machine and Cloud Service Sizes for Azure](http://msdn.microsoft.com/library/azure/dn197896.aspx).

### NTFS allocation unit size

NTFS volumes use a default cluster size of 4 KB. Based on our performance tests, we recommend changing the default cluster size to 64 KB during volume creation for both single disk and multiple disks (storage spaces) volumes.

## Data compression for I/O bound workloads

Some I/O intensive workloads can gain performance benefits through data compression. Compressed tables and indexes means more data stored in fewer pages, and hence require reading fewer pages from disk, which in turn can improve the performance of workloads that are I/O intensive.

For a data warehouse workload running on SQL Server in Azure VM, we found significant improvement in query performance by using page compression on tables and indexes, as shown in Figure 1.

**Figure 1: Query Performance with Data Compression**

Figure 1 compares performance of one query with no compression (NONE) and page compression (PAGE). As illustrated, the logical and physical reads are significantly reduced with page compression, and so is the elapsed time. As expected, CPU time of the query does go up with page compression, because SQL Server needs to decompress the data while returning results to the query. Your results will vary, depending upon your workload.

For an OLTP workload, we observed significant improvements in throughput (as measured by business transactions per second) by using page compression on selected tables and indexes that were involved in the I/O intensive workload. Figure 2 compares the throughput and CPU usage for the OLTP workload with and without page compression.

**Figure 2: OLTP Throughput and CPU Usage with Data Compression**

Note that you may see different results when you test your workloads in Azure Virtual Machine environment. But we recommend that you test data compression techniques for I/O intensive workloads and then decide which tables and indexes to compress. For more information, see [Data Compression: Strategy, Capacity Planning and Best Practices](http://msdn.microsoft.com/library/dd894051.aspx).

## Restore performance – instant file initialization

For databases of any significant size, enabling instant file initialization can improve the performance of some operations involving database files, such as creating a database or restoring a database, adding files to a database or extending the size of an existing file, autogrow, and so on. For information, see [How and Why to Enable Instant File Initialization](http://blogs.msdn.com/b/sql_pfe_blog/archive/2009/12/23/how-and-why-to-enable-instant-file-initialization.aspx).

To take advantage of instant file initialization, you grant the SQL Server (MSSQLSERVER) service account with SE\_MANAGE\_VOLUME\_NAME and add it to the **Perform Volume Maintenance Tasks** security policy. If you are using a SQL Server platform image for Azure, the default service account (NT Service\MSSQLSERVER) isn’t added to the **Perform Volume Maintenance Tasks** security policy. In other words, instant file initialization is not enabled in a SQL Server Azure platform image.

After adding the SQL Server service account to the **Perform Volume Maintenance Tasks** security policy, restart the SQL Server service.

The following figure illustrates observed test results for creating and restoring a 100 GB database with and without instant file initialization.

**Figure 3: Performance Impact of Instant File Initialization**

For more information, see [Database File Initialization](http://msdn.microsoft.com/en-us/library/ms175935(v=sql.105).aspx).

## Other existing best practices

Many of the best practices when running SQL Server on premises are still relevant in Azure Virtual Machines, including:

* **Limit or disable autogrow on the database:** **Autogrow is considered to be merely a contingency for unexpected growth. Do not manage your data and log growth on a day-to-day basis with autogrow. If autogrow is used, pre-grow the file using the Size switch.**
* **Disable autoshrink on the database: Make sure autoshrink is disabled to avoid unnecessary overhead that can negatively affect performance. For more information about autogrow and autoshrink, see** [Considerations for the "autogrow" and "autoshrink" settings in SQL Server](http://support.microsoft.com/kb/315512)**.**
* **Establish locked pages to reduce IO and any paging activities:** Lock pages in memory is a Windows policy that determines, which account can use a process to keep memory allocations pinned in physical memory. It prevents the system from paging the data to virtual memory on disk. When the SQL Server service account is granted this user right, buffer pool memory cannot be paged out by Windows. For more information about enabling the **Lock pages in memory** user right, see [How to: Enable the Lock Pages in Memory Option (Windows)](http://msdn.microsoft.com/en-us/library/ms190730.aspx).

# Performance troubleshooting fundamental concepts

This section provides an overview of how you troubleshoot SQL Server when running on a Azure virtual machine.

## Traditional factors that govern performance in SQL Server

Performance analysis in SQL Server is well documented. The [Troubleshooting Performance Problems in SQL Server 2008 article](http://msdn.microsoft.com/en-us/library/dd672789(v=sql.100).aspx) provides comprehensive information on this subject. You can also find other advanced performance-related blogs and articles on topics such as latch and spinlock analysis at [SQLCAT.com](http://sqlcat.com/). Here, let’s summarize the main performance factors:

* **Plan change/plan choice issues:** 
  + SQL Server Query Optimizer searches and chooses the optimal plan that can improve the system performance. The query optimizer’s choices are usually correct, but there are cases where it can be suboptimal. This can be caused by a number of factors, including out-of-date input data (such as statistics), poor index coverage and, in some cases, issues with the optimizer model or functionality. Query tuning and appropriate index creation and management can help you tune your I/O, CPU, and memory usage and optimize your workload performance.
  + As discussed previously, using SQL Server in Azure virtual machine has also other performance implications for the performance of your workload. Due to its multi-tenant nature and various capping and throttling mechanisms, you might notice different I/O performance results over time. That’s why we recommend that you minimize the number of reads and writes that your query requires by tuning the query plan, and also apply SQL Server performance optimization techniques, such as compression. For more information about the Optimizer, see the Optimizer chapter in the book *SQL Server 2008 Internals* or the upcoming *SQL Server 2012 Internals*.
* **Improperly configured software or hardware:** One common cause of I/O bottleneck is when I/O intensive files, such as data and log files for a high throughput database, are placed on the operating system disk drive. As a general rule, we recommend that you should add more data disks and balance your data and log files according to your IOPS requirements when using SQL Server in Azure Virtual Machines.
* **Locking and latching:** Excessive locking and latching can be caused by plan choice, or concurrency volumes due to overall system throughput; or certain schema and workload patterns. **Locking** is used to allow multiple users to make consistent changes to the same data in a controlled manner. **Latching** is used to adjudicate multi-user access to structures within SQL Server. Latching is somewhat orthogonal to locking; both issues can apply and the core resolution is often similar. If there is a hot latch or lock, the typical answer is to change the calling pattern (either by forcing different plan shapes or rearranging the calling code) to mitigate the impact of the blocking latch or lock on overall throughput. If you observe high PAGEIOLATCH waits on your system, this means that a user requests a page that is not in the buffer pool and I/O system is slow to respond. In this case, we recommend that you spread your I/O workload across more data disks or gain more memory by increasing the virtual machine instance size to improve the performance within Azure virtual machine.
* **Multi-user operations and system resource usage:** Some user operations may not run, or they may run at a reduced level of performance because of insufficient system resources. Some type of maintenance and schema management operations can increase resource utilization. For example, index rebuild operations can cause increased I/O workload on your system. You may think that your application’s queries or operations are taking longer than usual. But the internal cause is pressure on one or more resources. We recommend that you consider the resource usage of both the user application’s queries and the maintenance tasks in your capacity planning. Especially for I/O intensive operations, we recommend that you choose your virtual machine size appropriately and plan your maintenance windows to avoid your application’s peaks periods.
* **Checkpoints and system operations:** Flushing I/Os to disk during the checkpoint process can cause a spike in I/O operations. This might slow the performance of queries and impact throughput. It is important to determine your I/O workload and test for a period longer than the checkpoint frequency.

## Factors that govern performance for SQL Server in Azure Virtual Machines

Although the SQL Server binaries running in a traditional on-premises and Microsoft Azure virtual machine environments are identical, there are some infrastructure differences that affect the way that applications perform in SQL Server in Azure Virtual Machines compared to an on-premises dedicated enterprise server. An analogy that can be used to compare the experience of running SQL Server in Azure virtual machine is an on-premises virtualized SQL Server environment with dedicated virtualized resources (such as processor and memory) but no Hyper-V over-commit of memory or processors. In addition, the I/O performance varies in Microsoft Azure storage as it is a managed shared disk environment and performance at a point in time depends on other tenants’ activity and overall system load; this can happen in shared on-premises storage environments as well.

All the traditional SQL Server performance factors described in the previous section are still applicable when running SQL Server in Azure virtual machine environment. The following table summarizes most common resource bottleneck issues and provides a list of actions that you can take to resolve them.

|  |  |  |
| --- | --- | --- |
| **Issue** | Key performance indicators (KPIs) to monitor | **Actions to consider** |
| SQL Server CPU at or near 80% | % Processor Time (\_Total)  SOS\_SCHEDULER\_YIELD waits | * Increase number of CPUs by increasing your SQL Server virtual machine instance size (if possible). * Identify top consuming queries and tune. * Split out workload (for example, move a database off the SQL Server instance). |
| Near I/O capacity limits or  I/O latency increases | Average disk reads per second  Average disk writes per second  Disk reads per second  Disk writes per second  io\_virtual\_file\_stats  PAGEIOLATCH waits  SQL Server: Buffer Manager\Page Life Expectancy | * Check Page Life Expectancy counter value is low (<300 seconds). This can indicate memory pressure on the system is causing increased disk IO. Consider increasing instance size (if possible). * Identify which database and log files have I/O bottleneck. * Add more data disks and separate data files if you are at or near IOPS limits per disk.   **Note:** This can apply to any user created or **tempdb** databases.   * Tune queries to reduce reads and writes. * Consider enabling row or page compression to reduce the number of I/Os. |
| Memory resource pressure | Memory: Available Bytes  Memory: Pages per second  SQL Server: Buffer Manager\Page Life Expectancy  Process: Working Set (for SQL Server)  RESOURCE\_SEMAPHORE waits | * Check **max server memory** setting for SQL Server. * Increase memory by Increasing instance size or use high memory instance if possible. * Check which component of SQL Server utilizes memory (such as, CLR, high memory grants for application queries, and so on) and tune appropriately. |

As described in the table, you can resolve performance issues by following different approaches and actions. In traditional on-premises environment, you might prefer adding or purchasing more hardware resources to alleviate performance e problems. In Azure environment, the resources that are available per machine are smaller in number and less powerful than the typical on-premises enterprise servers. For example, while adding a data disk may increase disk throughput by 25 percent; tuning query workloads might reduce the overall I/O requirement by 90 percent in some cases. Therefore we recommend that you always follow a systematic approach that involves analyzing, tuning and redesigning to achieve better performance results.

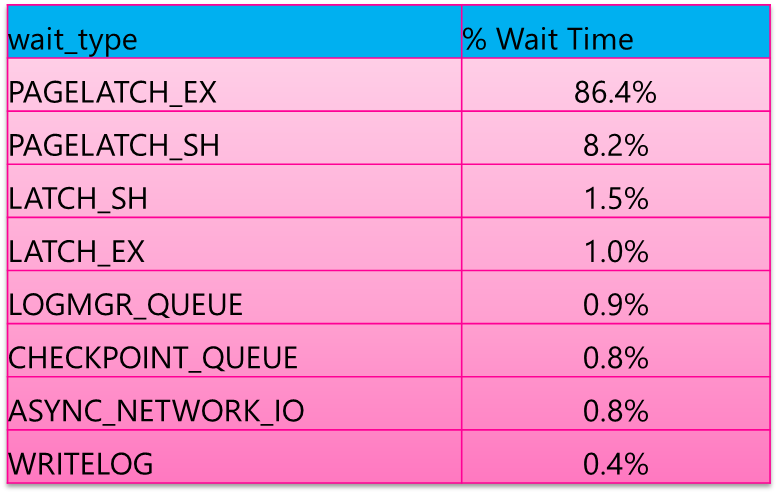
Optimizing your application for Azure Virtual Machine environment will provide valuable cost benefits because you can achieve a higher density per unit of compute. Unlike the traditional on-premises environment, Azure allows you reduce the number and size virtual machines immediately to reduce the operational costs. In addition, you can dynamically re-balance the size of machines based on seasonal usage peaks.

It is important that you develop a systematic monitoring and troubleshooting methodology to effectively run your SQL Server in Azure Virtual Machine environment.

## Performance monitoring methodology for SQL Server in Azure Virtual Machine

This section explains the approaches that can be used to identify and isolate performance issues while running SQL Server in Azure Virtual Machine environment:

* **Define key performance indicators (KPIs) to monitor resource utilization:** We recommend that you define KPIs for SQL Server and each important application tier. These should include [Windows Performance Monitor](http://technet.microsoft.com/library/cc749249.aspx) (Performance Monitor) counters for your application’s important tiers and components as well as SQL Server. In addition, you should monitor SQL Server’s performance related dynamic management views (DMVs) to identify the underlying causes of performance problems. We recommend that you define five to eight KPIs for each major application entity or component, such as SQL Server, application-related counters, and caching component counters. For more information, see the [Guidance on defining key performance indicators](#_Guidance_on_defining) section in the [Appendix](#_Appendix).
* **Monitor your KPIs to track resource utilization:** We recommend that you track and monitor KPIs by using tools such as Performance Monitor and SQL Server Management Studio. If application requests increase by a certain percent capacity, the proportional increase in the underlying system resources depends on how uniform the workload is. Make sure you track a latency metric, particularly for web applications running on SQL Server in Azure virtual machine.
* **Examine trends and patterns to identify issues as the workload increases:** As the number of users that are using your application increases, some system resources (such as processor, I/O, memory, and network) might become under pressure. For example, if the SQL Server workload reaches to its sustainable I/O limits, it becomes I/O bound. When this happens, the latency of the I/O subsystem increases first. Therefore, you may notice a corresponding increase on individual query execution times, which increase latency for end users. Secondly, throughput, such as the number of concurrent user queries that SQL Server can support, begins to level-off or decrease. Finally, if the pressure on a specific resource increases, you may notice unavailability issues, such as query and application timeouts. By monitoring Performance Monitor and the [sys.dm\_os\_wait\_stats](http://msdn.microsoft.com/library/ms179984.aspx) DMV, you can identify the potential performance problems before end users notice them. For example, this scenario would initially cause an increase in disk response time as measured by the logical disk performance counters, and an increase in the number of **PAGEIOLATCH** or log related waits that SQL Server provides.
* **Monitor DMVs to determine what resources your application is competing and waiting for:** 
  + **Monitor and identify instance level waits by** using [sys.dm\_os\_wait\_stats](http://msdn.microsoft.com/library/ms179984.aspx). Make sure to review the total wait time as a percentage of the total. The following table describes the wait profile that are indicative of Page Latch contention.

****

Other common profiles depend on the predominant wait type and include: **SOS\_SCHEDULER\_YIELD,** which indicates that the operation is waiting on the CPU to become free; long **PAGEIOLATCH** waits, which indicate that SQL Server is generating I/O requests faster than the I/O system can process them; and **RESOURCE\_SEMAPHORE, which indicates memory pressure on memory**. See the [Snapshot wait stats](#_Snapshot_wait_stats) script provided in the Appendix. It demonstrates how to automate the calculation of waits during a time interval. This is useful for identifying the dominant underlying wait in SQL Server when symptoms appear with your workload.

* + **Monitor query resource consumers** by using [sys.dm\_exec\_query\_stats](http://msdn.microsoft.com/library/ms189741.aspx) to identify top resource consumers and monitor query execution and efficiency
  + **Monitor I/O consumptions and characteristics** byusing [sys.dm\_io\_virtual\_file\_stats](http://msdn.microsoft.com/library/ms190326.aspx) and the logical disk Performance Monitor counters. The following table summarizes the key performance counters to monitor.

|  |  |  |
| --- | --- | --- |
| **Logical disk counter** | Typical storage term | **Suggested actions in**  **Microsoft Azure virtual machine environment** |
| Disk reads / second  Disk writes / second | IOPS | Measure the number of I/O’s per second.  Consider adding more data disks in line with your IOPS requirements. |
| Average disk sec / read  Average disk sec / write | Latency | Measure disk latency.  Note: Numbers might vary; look at averages over time. |
| Average disk bytes / read  Average disk bytes / write | Block size | Measure the size of I/O’s being issued.  Note: Larger I/O’s tend to have higher latency, such as those associated with BACKUP/RESTORE. |
| Average / current disk queue length | Outstanding or waiting IOPS | Provides insight into the applications I/O pattern. |
| Disk read bytes/sec  Disk write bytes/sec | Throughput or aggregate throughput | Measure of total disk throughput.  Note: Ideally, larger block scans should be able to heavily utilize connection bandwidth (for example, your throughput can be higher with a smaller number of larger IOPS). |

* + **Take snapshots of currently executing SQL Server requests** by using [sys.dm\_exec\_requests](http://msdn.microsoft.com/library/ms177648(v=sql.110).aspx) to check for locking, blocking, latching and other performance issues caused by resource contention related performance issues.
  + **Monitor the application and SQL Server event logs** to identify errors.
* **Use a delta approach to monitor beyond DMVs:** Some DMVs provide cumulative information from the last time that the SQL Server process was started, such as sys.dm\_os\_wait\_stats. Others contain a snapshot from a point in time, such as sys.dm\_exec\_requests. The performance of a query is affected by factors including plan selection and resource availability. An effective approach that works in on-premises and Azure Virtual Machine environments is to combine the usage of sys.dm\_os\_wait\_stats and sys.dm\_exec\_query\_stats. This helps you to understand the query performance and resource constraints, which can be inferred from the system-wait information. To identify locking and blocking issues, you should routinely monitor active request information using sys.dm\_exec\_requests. In summary, we recommend that you:
  + Take a regular periodic snapshots of query stats, wait stats, and exec requests.
  + Calculate the delta between two snapshots to understand what happened during that period. You can use the sample script [Snapshot wait stats](#_Snapshot_wait_stats) in the Appendix for this purpose.
* **Monitor Spinlock and Backoff events:** Spinlocks are used to synchronize access to key memory regions that SQL Server uses for internal constructs. Use sys.dm\_os\_spinlock\_stats to monitor the number of spins. For more information, see [Diagnosing and Resolving Spinlock Contention on SQL Server](http://www.microsoft.com/en-us/download/details.aspx?id=26666) article.

# Appendix

The scripts and guidance covered in this section can be used to facilitate the information given in this article:

* **Guidance on defining key performance indicators (KPIs):** Provides information on KPIs that you can use to monitor application tier performance and user characteristics while running SQL Server in Azure.
* **Raw storage performance testing scripts:** Discusses the usage of SQLIO.
* **Snapshot wait stats script:** Demonstrates how to automate the calculation of waits during a time interval.
* **Requests executing on the system script:** Provides a view of requests executing on SQL Server sorted by total elapsed time, includes the corresponding plan for each statement.
* **Top query statements and plan by total CPU time script:** Provides information from Query Stats on the top overall resource consumers.
* **Snapshot spinlock stats script:** Uses a temporary table to provide the delta of spinlock information since the last execution of this script.
* **Snapshot I/O stats script:** Uses a temporary table to provide the delta of I/O information since the last execution of this script.

## Raw storage performance testing scripts

You can run the following SQLIO scripts in Azure Virtual Machine to generate some of the most common I/O patterns that SQL Server utilizes and measure related performance results on different storage configurations.

The following SQLIO test scripts demonstrate testing random 8K reads/writes (a typical OLTP I/O pattern), sequential writes for log files, and large sequential reads and writes for table scans and OLAP workloads.

The script uses the following options:

* + The **-k** option to specify the I/O operation type (read or write)
  + The **-s** option to specify the test duration in seconds
  + The **–f** option to specify the type of I/O access (sequential or random)
  + The **–o** option to specify the number of outstanding requests
  + The **–b** option to specify the size of the I/O request in bytesblock size
  + The **–LS** option to capture the disk latency option
  + The **–F** option to specify the name of the file which contain the test files to run SQLIO against

Copy and save the following script in a file called **exectests.bat**.

|  |
| --- |
| ::Test random 8K reads/writes |
| sqlio -kW -s300 -frandom -o32 -b8 -LS -Fparam.txt |
| sqlio -kR -s300 -frandom -o32 -b8 -LS -Fparam.txt |
| ::Test random 32K reads/writes (for example, SQL Server Analysis Services I/O pattern)  sqlio -kW -s300 -frandom -o32 -b32 -LS -Fparamnew.txt  sqlio -kR -s300 -frandom -o32 -b32 -LS -Fparamnew.txt |
| ::Test small sequential writes |
| sqlio -kW -s180 -fsequential -o1 -b4 -LS -Fparam.txt |
| sqlio -kW -s180 -fsequential -o1 -b8 -LS -Fparam.txt |
| sqlio -kW -s180 -fsequential -o1 -b16 -LS -Fparam.txt |
| sqlio -kW -s180 -fsequential -o1 -b32 -LS -Fparam.txt |
| sqlio -kW -s180 -fsequential -o1 -b64 -LS -Fparam.txt |
|  |
| ::Test large sequential reads/writes |
| sqlio -kR -s180 -fsequential -o8 –b8 -LS -Fparam.txt |
| sqlio -kR -s180 -fsequential -o8 -b64 -LS -Fparam.txt |
| sqlio -kR -s180 -fsequential -o8 -b128 -LS -Fparam.txt |
| sqlio -kR -s180 -fsequential -o8 -b256 -LS -Fparam.txt |
| sqlio -kR -s180 -fsequential -o8 –b512 -LS -Fparam.txt |
| sqlio -kW -s180 -fsequential -o8 –b8 -LS -Fparam.txt |
| sqlio -kW -s180 -fsequential -o8 -b64 -LS -Fparam.txt |
| sqlio -kW -s180 -fsequential -o8 -b128 -LS -Fparam.txt |
| sqlio -kW -s180 -fsequential -o8 -b256 -LS -Fparam.txt |
| sqlio -kW -s180 -fsequential -o8 –b512 -LS -Fparam.txt |

The following is a copy of the param.txt configuration file that we referenced in our test scripts. Basically, the test scripts run a series of tests against the drive or drives specified in the param.txt file. This file should reside in the same directory as SQLIO.exe. The options on each line of the param.txt file are as follows, where **0x0** is a mask value:

***PathToTestFile NumberofThreadsPerTestFile 0x0 TestFileSizeinMegaBytes***

This param.txt file skips operating system and temporary disks (**C:** and **D:**) and tests a 16 data disks volume (F:) (using 1 thread per disk), with a file size of 50 GB.

|  |
| --- |
| #c:\testfile.dat 2 0x0 100 |
| #d:\testfile.dat 1 0x0 1000 |
| f:\testfile.dat 16 0x0 50000 |

Open a Command Prompt window, then run the test batch file as follows.

**exectests.bat > results.txt**

This operation captures all test results in a text file that can be processed manually or automatically to extract relevant disk performance figures later.

## Guidance on defining key performance indicators

We recommend that you define key performance indicators (KPIs) for all of the components in your application scenario. Typically, you can use performance counters to define your KPIs.

This section lists KPIs that we’ve used to monitor application tier performance and user characteristics while running SQL Server in Azure. Note that the KPIs listed in this section are meant to be guidance only. You should define your KPIs based on the characteristics of your application’s components.

**Typical SQL Server KPIs:**

* Maximum value for \Process(SQLServ)\% Processor Time
* Average value for \Process(SQLServ)\% Processor Time
* Maximum value for \Processor(\_Total)\% Processor Time
* Average value for \Processor(\_Total)\% Processor Time
* Maximum value for \SQLServer:SQL Statistics\Batch Requests/sec
* Average value for \SQLServer:SQL Statistics\Batch Requests/sec

If your workload is likely to be I/O bound, you should also add the Logical Disk Performance Monitor counters referenced earlier. If you use additional SQL Server features, such as AlwaysOn, it is recommended that you add the appropriate performance monitor counters.

**Typical web application tier KPIs:**

* Maximum value for \ASP.NET Applications (\_Total\_)\Requests/sec
* Average value for \ASP.NET Applications (\_Total\_)\Requests/sec
* Average value for \Memory\Available Mbytes
* Maximum value for \Processor(\_Total)\% Processor Time
* Average value for \Processor(\_Total)\% Processor Time
* Average value for \ASP.NET\Request Wait Time
* Average value for \ASP.NET\Request Execution Time
* Average value for \ASP.NET\Requests Queued
* Average value for \ASP.NET\Requests Rejected
* Average value for \ASP.NET\Requests Current

**Typical user/test characteristics KPI:**

* Number of concurrent users
* Average/Max request execution time
* Number of web servers
* Ramp up period, test method
* Start and end time of test

### How to use KPIs

First, identify five to eight KPIs for each application component, such as SQL Server, application tier, and a similar number for the test characteristics KPI’s. The KPIs you choose should be a subset of overall performance counter collection. Note that these KPIs represent an overall performance of the application during different time intervals. In a production environment, you may notice KPIs return different information during the day. We recommend that you calculate and analyze KPIs regularly by using tools, such as Excel. For example, you should be able to make assertions such as “When we scale from one application server to two application servers, the CPU utilization on my SQL Server increases by 2.5x”. In this example, you could continue investigating further using detailed Performance Monitor logs and DMV stats to understand why SQL Server CPU utilization has increased by 2.5x and understand whether this is a normal characteristic of our workload or there is an issue we need to investigate.

You can automate the collection of the raw performance logs and the calculation of the KPIs by using the [Logman](http://technet.microsoft.com/en-us/library/cc753820(v=WS.10).aspx) and [Log Parser](http://technet.microsoft.com/en-us/scriptcenter/dd919274.aspx) tools.

## SQL Server troubleshooting scripts

### Snapshot wait stats script

/\* Snapshot the current wait stats and store them so that they can be compared over a time period

Return the statistics between this point in time and the last collection point in time.

\*/

use tempdb

go

if exists (select \* from sys.objects where name = 'snap\_waits')

drop procedure snap\_waits

go

create procedure snap\_waits

as

declare @current\_snap\_time datetime

declare @previous\_snap\_time datetime

set @current\_snap\_time = GETDATE()

if not exists(select name from tempdb.sys.sysobjects where name like 'wait\_stats%')

create table wait\_stats

(

wait\_type varchar(128)

,waiting\_tasks\_count bigint

,wait\_time\_ms bigint

,avg\_wait\_time\_ms int

,max\_wait\_time\_ms bigint

,signal\_wait\_time\_ms bigint

,avg\_signal\_wait\_time int

,snap\_time datetime

)

insert into wait\_stats (

wait\_type

,waiting\_tasks\_count

,wait\_time\_ms

,max\_wait\_time\_ms

,signal\_wait\_time\_ms

,snap\_time

)

select

wait\_type

,waiting\_tasks\_count

,wait\_time\_ms

,max\_wait\_time\_ms

,signal\_wait\_time\_ms

,@current\_snap\_time

from sys.dm\_os\_wait\_stats

--get the previous collection point

select top 1 @previous\_snap\_time = snap\_time from wait\_stats

where snap\_time < (select max(snap\_time) from wait\_stats)

order by snap\_time desc

--get delta in the wait stats

select top 10

s.wait\_type

, (e.wait\_time\_ms - s.wait\_time\_ms)/((e.waiting\_tasks\_count - s.waiting\_tasks\_count)) as [avg\_wait\_time\_ms]

,(e.signal\_wait\_time\_ms - s.signal\_wait\_time\_ms)/((e.waiting\_tasks\_count - s.waiting\_tasks\_count)) as [avg\_signal\_time\_ms]

, (e.waiting\_tasks\_count - s.waiting\_tasks\_count) as [waiting\_tasks\_count]

, (e.wait\_time\_ms - s.wait\_time\_ms) as [wait\_time\_ms]

, (e.max\_wait\_time\_ms) as [max\_wait\_time\_ms]

, (e.signal\_wait\_time\_ms - s.signal\_wait\_time\_ms) as [signal\_wait\_time\_ms]

, s.snap\_time as [start\_time]

, e.snap\_time as [end\_time]

, DATEDIFF(ss, s.snap\_time, e.snap\_time) as [seconds\_in\_sample]

from wait\_stats e

inner join (

select \* from wait\_stats

where snap\_time = @previous\_snap\_time

) s on (s.wait\_type = e.wait\_type)

where

e.snap\_time = @current\_snap\_time

and s.snap\_time = @previous\_snap\_time

and e.wait\_time\_ms - s.wait\_time\_ms > 0

and e.waiting\_tasks\_count - s.waiting\_tasks\_count > 0

and e.wait\_type NOT IN ('LAZYWRITER\_SLEEP', 'SQLTRACE\_BUFFER\_FLUSH'

, 'SOS\_SCHEDULER\_YIELD' ,'DBMIRRORING\_CMD', 'BROKER\_TASK\_STOP'

, 'CLR\_AUTO\_EVENT', 'BROKER\_RECEIVE\_WAITFOR', 'WAITFOR'

, 'SLEEP\_TASK', 'REQUEST\_FOR\_DEADLOCK\_SEARCH', 'XE\_TIMER\_EVENT'

, 'FT\_IFTS\_SCHEDULER\_IDLE\_WAIT', 'BROKER\_TO\_FLUSH', 'XE\_DISPATCHER\_WAIT'

, 'SQLTRACE\_INCREMENTAL\_FLUSH\_SLEEP')

order by (e.wait\_time\_ms - s.wait\_time\_ms) desc

--clean up table

delete from wait\_stats

where snap\_time < @current\_snap\_time

go

exec snap\_waits

### Requests executing on the system script

/\* requests executing on the system

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

select r.session\_id

,blocking\_session\_id

,wait\_type

,wait\_time

,wait\_resource

,r.status

,r.cpu\_time

,r.total\_elapsed\_time

,r.reads

,s.reads [session reads]

,s.logical\_reads [session logical reads]

,r.writes

,r.logical\_reads

--,r.scheduler\_id

,s.host\_name

,qt.dbid

,qt.objectid

,substring(substring(qt.text,r.statement\_start\_offset/2+1,

(case when r.statement\_end\_offset = -1

then len(convert(nvarchar(max), qt.text)) \* 2

else r.statement\_end\_offset end - r.statement\_start\_offset)/2)

, 1, 255) --substring

as statement\_text

--,qp.query\_plan

from sys.dm\_exec\_requests r

inner join sys.dm\_exec\_sessions s on (s.session\_id = r.session\_id)

cross apply sys.dm\_exec\_sql\_text(sql\_handle) as qt

cross apply sys.dm\_exec\_query\_plan (plan\_handle) as qp

where r.session\_id > 50

--and r.session\_id = 55

order by r.total\_elapsed\_time desc --r.status -- r.scheduler\_id, r.status, r.session\_id

### Top query statements and plan by total CPU time

/\* Top statements by total CPU time

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SELECT TOP 25

SUBSTRING(qt.text,qs.statement\_start\_offset/2+1,

(case when qs.statement\_end\_offset = -1

then len(convert(nvarchar(max), qt.text)) \* 2

else qs.statement\_end\_offset end -qs.statement\_start\_offset)/2)

as statement\_text,

--substring (qt.text , 1, 512) as batch\_text,

qs.total\_worker\_time/qs.execution\_count as average\_cpu\_time,

qs.total\_elapsed\_time/qs.execution\_count as average\_elapsed\_time,

qs.total\_logical\_reads/qs.execution\_count as average\_logical\_reads,

qs.total\_logical\_writes/qs.execution\_count as average\_logical\_writes,

qs.execution\_count,

qs.plan\_generation\_num,

qs.total\_worker\_time,

qs.total\_elapsed\_time,

cast((cast(qs.total\_worker\_time as decimal) / cast(qs.total\_elapsed\_time as decimal) \* 100) as int) as cpu\_vs\_elapsed\_percentage,

qs.total\_logical\_reads,

qs.total\_logical\_writes,

db\_name(qt.dbid) as [database name],

qs.plan\_handle,

qt.objectid

,qp.query\_plan

FROM sys.dm\_exec\_query\_stats qs

cross apply sys.dm\_exec\_sql\_text(qs.sql\_handle) as qt

cross apply sys.dm\_exec\_query\_plan(qs.plan\_handle) as qp

--order by qs.total\_logical\_reads/qs.execution\_count desc

--ORDER BY total\_logical\_reads desc

--ORDER BY total\_elapsed\_time desc

ORDER BY total\_worker\_time desc

### Snapshot spinlock stats

--115982764 (with lock)

--31250708

/\* Snapshot the current spinlock stats and store so that this can be compared over a time period

Return the statistics between this point in time and the last collection point in time.\*/

use tempdb

go

if exists (select \* from sys.objects where name = 'snap\_spins')

drop procedure snap\_spins

go

create procedure snap\_spins

as

declare @current\_snap\_time datetime

declare @previous\_snap\_time datetime

set @current\_snap\_time = GETDATE()

if not exists(select name from tempdb.sys.sysobjects where name like 'spin\_waits%')

create table spin\_waits

(

lock\_name varchar(128)

,collisions bigint

,spins bigint

,sleep\_time bigint

,backoffs bigint

,snap\_time datetime

)

--capture the current stats

insert into spin\_waits

(

lock\_name

,collisions

,spins

,sleep\_time

,backoffs

,snap\_time

)

select name

,collisions

,spins

,sleep\_time

,backoffs

,@current\_snap\_time

from sys.dm\_os\_spinlock\_stats

select top 1 @previous\_snap\_time = snap\_time from spin\_waits

where snap\_time < (select max(snap\_time) from spin\_waits)

order by snap\_time desc

--get delta in the spin locks stats

select top 10

spins\_current.lock\_name

, (spins\_current.spins - spins\_previous.spins) as spins

,(spins\_current.backoffs - spins\_previous.backoffs) as backoffs

, (spins\_current.collisions - spins\_previous.collisions) as collisions

, (spins\_current.sleep\_time - spins\_previous.sleep\_time) as sleep\_time

, spins\_previous.snap\_time as [start\_time]

, spins\_current.snap\_time as [end\_time]

, DATEDIFF(ss, @previous\_snap\_time, @current\_snap\_time) as [seconds\_in\_sample]

from spin\_waits spins\_current

inner join (

select \* from spin\_waits

where snap\_time = @previous\_snap\_time

) spins\_previous on (spins\_previous.lock\_name = spins\_current.lock\_name)

where

spins\_current.snap\_time = @current\_snap\_time

and spins\_previous.snap\_time = @previous\_snap\_time

and spins\_current.spins > 0

order by (spins\_current.spins - spins\_previous.spins) desc

--clean up table

delete from spin\_waits

where snap\_time < @current\_snap\_time

go

exec snap\_spins

### Snapshot I/O stats

/\* Snapshot the current file stats and store them so that they can be compared over a time period

Return the statistics between this point in time and the last collection point in time.

This uses a persisted table in tempdb. After the needed data is captured, drop this table.

use tempdb

go

drop table \_iostats\_

\*/

use tempdb

go

if not exists(select name from tempdb.sys.sysobjects where name like '\_iostats\_')

create table \_iostats\_

(

database\_id int

,file\_id int

,file\_guid uniqueidentifier

,num\_of\_bytes\_read bigint

,num\_of\_bytes\_written bigint

,num\_of\_reads bigint

,num\_of\_writes bigint

,io\_stall\_write\_ms bigint

,io\_stall\_read\_ms bigint

,size\_on\_disk\_bytes bigint

,physical\_name nvarchar(260)

,type\_desc nvarchar(60)

,snap\_time datetime

)

declare @current\_snap\_time datetime

declare @previous\_snap\_time datetime

set @current\_snap\_time = GETDATE()

insert into \_iostats\_ (

database\_id

,file\_id

,file\_guid

,num\_of\_bytes\_read

,num\_of\_bytes\_written

,num\_of\_reads

,num\_of\_writes

,io\_stall\_write\_ms

,io\_stall\_read\_ms

,size\_on\_disk\_bytes

,physical\_name

,type\_desc

,snap\_time

)

select

vfs.database\_id

,vfs.file\_id

,mf.file\_guid

,vfs.num\_of\_bytes\_read

,vfs.num\_of\_bytes\_written

,vfs.num\_of\_reads

,vfs.num\_of\_writes

,vfs.io\_stall\_write\_ms

,vfs.io\_stall\_read\_ms

,vfs.size\_on\_disk\_bytes

,mf.physical\_name

,mf.type\_desc

,@current\_snap\_time

from sys.dm\_io\_virtual\_file\_stats(null, null) as vfs

join sys.master\_files as mf on vfs.database\_id = mf.database\_id and vfs.file\_id = mf.file\_id

where vfs.database\_id > 4 or vfs.database\_id = 2

order by vfs.database\_id, vfs.file\_id

select top 1 @previous\_snap\_time = snap\_time from \_iostats\_

where snap\_time < (

select max(snap\_time) from \_iostats\_) order by snap\_time desc

print 'Current snap time: ' + convert(varchar(32), @current\_snap\_time)

print 'Previous snap time: ' + convert(varchar(32), @previous\_snap\_time)

declare @tick\_count\_between\_snaps bigint

set @tick\_count\_between\_snaps = DATEDIFF(ms, @previous\_snap\_time, @current\_snap\_time)

print '@tick\_count\_between\_snaps: ' + convert(varchar(32), @tick\_count\_between\_snaps)

select

time\_in\_sample\_secs = DATEDIFF(s, @previous\_snap\_time, @current\_snap\_time)

,db = db\_name(iostats\_now.database\_id)

,iostats\_now.physical\_name

,iostats\_now.type\_desc

,iostats\_now.file\_id

,(iostats\_now.num\_of\_bytes\_read - iostats\_lastsnap.num\_of\_bytes\_read) num\_of\_bytes\_read

,(iostats\_now.num\_of\_bytes\_written - iostats\_lastsnap.num\_of\_bytes\_written) num\_of\_bytes\_written

,(iostats\_now.num\_of\_reads - iostats\_lastsnap.num\_of\_reads) num\_of\_reads

,(iostats\_now.num\_of\_writes - iostats\_lastsnap.num\_of\_writes) num\_of\_writes

,avg\_read\_IOPs = case when (iostats\_now.num\_of\_reads - iostats\_lastsnap.num\_of\_reads) = 0 then 0 else ((iostats\_now.num\_of\_reads - iostats\_lastsnap.num\_of\_reads) /(@tick\_count\_between\_snaps/1000)) end

,avg\_read\_bytes\_sec = case when (iostats\_now.num\_of\_bytes\_read - iostats\_lastsnap.num\_of\_bytes\_read) = 0 then 0 else ((iostats\_now.num\_of\_bytes\_read - iostats\_lastsnap.num\_of\_bytes\_read)/(@tick\_count\_between\_snaps/1000)) end

,avg\_read\_stall\_ms = case when (iostats\_now.num\_of\_reads - iostats\_lastsnap.num\_of\_reads) = 0 then 0 else ((iostats\_now.io\_stall\_read\_ms - iostats\_lastsnap.io\_stall\_read\_ms) /(iostats\_now.num\_of\_reads - iostats\_lastsnap.num\_of\_reads)) end

,avg\_read\_size = case when (iostats\_now.num\_of\_reads - iostats\_lastsnap.num\_of\_reads) = 0 then 0 else ((iostats\_now.num\_of\_bytes\_read - iostats\_lastsnap.num\_of\_bytes\_read)/(iostats\_now.num\_of\_reads - iostats\_lastsnap.num\_of\_reads)) end

,avg\_write\_IOPs = case when (iostats\_now.num\_of\_writes - iostats\_lastsnap.num\_of\_writes) = 0 then 0 else ((iostats\_now.num\_of\_writes - iostats\_lastsnap.num\_of\_writes) /(@tick\_count\_between\_snaps/1000)) end

,avg\_write\_bytes\_sec = case when (iostats\_now.num\_of\_bytes\_written - iostats\_lastsnap.num\_of\_bytes\_written) = 0 then 0 else ((iostats\_now.num\_of\_bytes\_written - iostats\_lastsnap.num\_of\_bytes\_written)/(@tick\_count\_between\_snaps/1000)) end

,avg\_write\_stall\_ms = case when (iostats\_now.num\_of\_writes - iostats\_lastsnap.num\_of\_writes) = 0 then 0 else ((iostats\_now.io\_stall\_write\_ms - iostats\_lastsnap.io\_stall\_write\_ms) /(iostats\_now.num\_of\_writes - iostats\_lastsnap.num\_of\_writes)) end

,avg\_write\_size = case when (iostats\_now.num\_of\_writes - iostats\_lastsnap.num\_of\_writes) = 0 then 0 else ((iostats\_now.num\_of\_bytes\_written - iostats\_lastsnap.num\_of\_bytes\_written)/(iostats\_now.num\_of\_writes - iostats\_lastsnap.num\_of\_writes)) end

,iostats\_now.size\_on\_disk\_bytes

,filegrowth = iostats\_now.size\_on\_disk\_bytes - iostats\_lastsnap.size\_on\_disk\_bytes

,iostats\_now.file\_guid

from \_iostats\_ as iostats\_now

inner join

(select \* from \_iostats\_

where snap\_time = @previous\_snap\_time)

iostats\_lastsnap on ( --iostats\_lastsnap.file\_guid = iostats\_now.file\_guid

iostats\_lastsnap.file\_id = iostats\_now.file\_id AND iostats\_lastsnap.database\_id = iostats\_now.database\_id

)

where (iostats\_now.database\_id > 4 or iostats\_now.database\_id = 2)

and iostats\_now.snap\_time = @current\_snap\_time

and iostats\_lastsnap.snap\_time = @previous\_snap\_time

order by iostats\_now.database\_id asc, iostats\_now.file\_id asc

--clean up

delete from \_iostats\_

where snap\_time = @previous\_snap\_time

--drop table \_iostats\_

### Performance monitor

In addition to the scripts provided earlier, you should capture the following performance monitor counters over the same time period.

\LogicalDisk(\*)\Avg. Disk Bytes/Read

\LogicalDisk(\*)\Avg. Disk Bytes/Write

\LogicalDisk(\*)\Avg. Disk Queue Length

\LogicalDisk(\*)\Avg. Disk sec/Read

\LogicalDisk(\*)\Avg. Disk sec/Write

\LogicalDisk(\*)\Current Disk Queue Length

\LogicalDisk(\*)\Disk Read Bytes/sec

\LogicalDisk(\*)\Disk Reads/sec

\LogicalDisk(\*)\Disk Write Bytes/sec

\LogicalDisk(\*)\Disk Writes/sec

\Memory\Available MBytes

\Memory\Free System Page Table Entries

\Memory\Pages/sec

\Network Interface(\*)\\*

\Process(\*)\\*

\Processor(\*)\% Privileged Time

\Processor(\*)\% Processor Time

\SQLServer:Availability Replica(\*)\\*

\SQLServer:Access Methods\\*

\SQLServer:Buffer Manager\\*

\SQLServer:Buffer Node\\*

\SQLServer:Databases(\*)\\*

\SQLServer:Database Replica(\*)\\*

\SQLServer:General Statistics\\*

\SQLServer:Latches\\*

\SQLServer:Locks(\_Total)\Average Wait Time (ms)

\SQLServer:Locks(\_Total)\Lock Requests/sec

\SQLServer:Memory Manager\\*

\SQLServer:Plan Cache\\*

\SQLServer:Wait Statistics\\*

\SQLServer:SQL Statistics\\*

\System\Context Switches/sec

\System\Processor Queue Length

Note that we also recommend that you capture all replication counters when using replication.

The **logman utility** can be used to create the Performance Monitor counter set. Simply copy the above counter definition listed here into a file, such as counters.txt, and then run the following at the command prompt:

**logman create counter <counterfilename> –cf counters.txt**