Violin MEMORY

SQL Server Best Practices Guide

Deployment and Tuning Recommendations for Microsoft® SQL Server®

Document Number: 535-0248-00 Rev 01
January 2016

LEGAL NOTICE

Violin Memory, Inc.
4555 Great America Parkway
Santa Clara, CA 95054
USA

Table of Contents

1 Introduction

The purpose of this document is to present key best practices that Violin Memory engineers have found to be of benefit when configuring and using Violin arrays. The suggestions encompass SQL Server deployment and tuning configurations for SQL Server versions 2005 and forward. Any SQL version-specific recommendations will be highlighted in each section.

1.1 Intended Audience

This document is intended for SQL Server and Windows Server® administrators who have prior knowledge of and experience with SQL Server database installation and configuration in an enterprise-scale environment.

1.2 How to Use This Guide

This guide is designed to work in combination with the Windows Server and array version-specific best practices documents to provide full hardware-to-software coverage. The recommendations in this document focus on the specific deployment and tuning best practices to achieve optimal performance and reliability at the Microsoft SQL Server and database level.

Some deployment and tuning recommendations interact with or affect options or configurations at the Windows or array level so it is suggested to review all three documents before finalizing a deployment configuration or architecture. Also, it is highly recommended that you test different options to make sure that these recommendations work optimally for your specific workloads.

1.3 Reference Documents

The following documents are available for download from the Violin Memory Support site at:
http://www.violin-memory.com/support/

- *7300 Flash Storage Platform Best Practices Guide*
- Best Practices for Windows
- Violin 6000 Series Best Practices

2 SQL Server Best Practices

The following recommendations focus on the SQL Server server-level configuration and tuning parameters as well as architectural options and considerations.

2.1 Trace Flags and Startup Parameters

There are a handful of trace flags and startup parameters that can be applied that can improve performance. But, flags and server-level parameters must be used with care because their impact can vary depending on the specific workload. While there are many trace flags, Violin Memory's recommendations are for those that effect performance. Also, be aware that these flags are global to

the SQL Server. The following are only recommendations. It is recommended that each flag be tested to determine its value and impact for each server.

Trace Flags:

652 Disables read ahead for the server. Only use for pure OLTP applications, small random reads.

661 Disables the ghost record removal process. A ghost record is the result of a delete operation. When SQL Server deletes a record, the deleted record is kept as a ghost record. Later, the deleted record is purged by the ghost record removal process. When this process is disabled, the deleted record is not purged. Therefore, the space that the deleted record consumes is not freed. This behavior affects space consumption and the performance of scan operations. SCOPE: Global. If you turn off this trace flag, the ghost record removal process works correctly.

834 Causes SQL Server to use Windows large-page allocations for the memory that is allocated for the buffer pool. Good for DSS applications, and in general reduces TLB thrashing, as there are fewer pages in memory to track.

8744 Disables pre-fetching for the Nested Loops operator. Incorrect use of this trace flag may cause additional physical reads when SQL Server executes plans that contain the Nested Loops operator.

Startup Parameters:

-E Increases the number of extents that are allocated for each file in a file group. This is good for DSS applications.

-K[n] Increases the size of the buckets in the Ring Buffer. [n] is in megabytes.

To configure startup options:

1. In SQL Server Configuration Manager, click **SQL Server Services**.
2. In the right pane, right-click **SQL Server** (<instance_name>), and then click **Properties**.
3. On the **Advanced** tab, in the **Startup Parameters** box, type the parameters separated by semicolons (;).
4. Click OK.
5. Restart the database engine

Example:

```
<…>;-T 834;-E
```

2.2 NUMA

With the proliferation of Non-Uniform Memory Architecture (NUMA) systems, SQL Server and Windows Server have been enhanced to recognize these architectures and take steps to optimize memory access.

By default, Windows Server will interrogate the underlying hardware to determine the NUMA topology, if present. SQL Server (since SQL Server 2005) will in turn interrogate Windows Server, which will pass on the NUMA topology information. SQL Server will attempt to optimize user and data locality on the

same NUMA node. This helps to minimize intra-node memory access and avoid the associated higher latencies.

SQL Server has taken NUMA handling a step further, with two additional features regarding NUMA that allow more control and greater benefit from this feature:

1. **Soft NUMA** – This allows artificial NUMA nodes to be created in the Windows Registry. The Soft NUMA nodes must be a subset of the hardware NUMA node, typically composed of all the cores in a given socket. When combined with connection affinity, the workload can be finely tuned to evenly divide the work through all cores in the system, or segment work between reporting and OLTP to different processors.
2. **Connection Affinity** – This feature gives the application the ability to connect to a specific NUMA or Soft NUMA node, which will force the thread code and data stack to the same NUMA node, minimizing intra-node memory accesses.

2.3 CPU Affinity

CPU/core affinity should be reviewed if any of these cases are true:

1. There are processes or applications other than SQL Server residing on the same server as SQL Server.
2. SQL Server is running inside of a virtual machine.
3. There is more than one SQL Server running on the host server.
4. The server is of hard NUMA architecture. (4+ sockets or 2+ IOH's. Refer to the host server's manual to determine if it contains more than one IOH.)

If any of these situations apply, the system architect should review the affinity settings of each process/application. In general, it is optimal to have dedicated resources (cores) per workload over having many workloads all share the same resources. This can lead to optimal memory and core utilization.

2.4 MAXDOP: Max Degree of Parallelism

Max degree of parallelism, also referred to as MAXDOP, controls the number of processes a session can start, i.e. parallelism, for a given task, such as index creation, query data processing, etc. The default configuration value for this is 0, which allows SQL Server to dynamically choose the degree of parallelization up to the number of logical processors.

MAXDOP used along with the "cost threshold for parallelism" parameter. The optimizer will "cost" the query and if the cost is greater than or equal to the "cost threshold for parallelism", the optimizer will then parallelize the query to a maximum number of threads determined by the MAXDOP. This is a dynamic setting (no restart required).

The database engine's logic and execution plans change and evolve from version to version and this is one of the most significant parameters to test and tune when upgrading versions and hardware. This is especially true when upgrading from a disk-based storage subsystem to a flash-based storage subsystem in which random reads become performance enhancing over performance degrading.

Commonly, upon upgrading from disk-based storage to flash-based storage, the MAXDOP parameter will need tuning to increase the number of threads and thereby decrease query run times.

In OTLP type applications I/O access is defined by small, atomic reads and writes that will not benefit from parallelization. A MAXDOP setting of 1 removes the ability to execute in parallel mode, which reduces the optimizer's number of options to consider and therefore reduces execution time. This can lead to faster OLTP transactions. The correct setting can only be determined by testing, but Violin recommends it not be left at the default of 0 for OLTP applications.

2.5 Max Worker Threads

Thread pooling helps optimize performance when large numbers of clients are connected to the server. Usually, a separate operating system thread is created for each query request. However, with hundreds of connections to the server, using one thread per query request can consume large amounts of system resources. The "max worker threads" option enables SQL Server to create a pool of worker threads to service a larger number of query requests, which improves performance. Setting this to 0 allows SQL to set the default on SQL Server startup, which is good for most systems. The actual formula takes into account the number or processors, workload, etc. Microsoft's recommendations for this setting can be found at: http://msdn.microsoft.com/en-us/library/ms190219.aspx

2.6 Lightweight Pooling

Lightweight pooling shifts some of the task-scheduling overhead from the OS kernel to a lighter overhead SQL function. This is also good for OLTP applications but has been known to cause issues with Distributed Transaction Coordinator (DTC). This is a dynamic option. This is useful in some specific, highly-tuned OLTP applications with high transaction loads, but in general should only be used if the specific application has been well tested with it and it has shown to be of benefit.

2.7 High Priority

This option increases the OS processing priority for SQL Server process (es). This is only good if you have multiple applications running on the same server. It is dynamic.

2.8 HA / DR: High Availability / Disaster Recovery

Violin Memory's 7000 Series Flash Storage Platforms (FSPs) offer a range of HA and DR features including snapshots, replication, mirroring and stretch clustering. Each feature comes with its own considerations; 7000 Series best practices documentation should be reviewed before choosing an appropriate HA or DR strategy.

To allow for optimal performance, 7000 Series FSPs allow for the usage of both gateways for servicing workloads. This means that some LUNs can be active on one gateway while other LUNs are active on the other gateway. A common use case would be to have data reduction LUNs on Memory Gateway A while servicing thick and thin LUNs on Memory Gateway B.

When choosing to deploy both gateways as active for individual workloads, be sure to allocate all LUNs associated with a database to only one gateway. This allows for technologies like snapshots and replication to coordinate multiple LUNs as a single group.

Violin Memory's snapshot technology takes snapshots at the LUN level. It is recommended that you architect the LUNs and volumes such that databases not needing to be snapped, or restored, together reside on separate LUNs and volumes.

3 Database Best Practices

This section focuses on the configuration, tuning and architectural considerations to be evaluated at the individual database level.

3.1 Separate Data, Log and tempdb

Windows will allocate one I/O handling thread per underlying LUN (not volume). This can both create a singled threaded type bottleneck in performance and also mix large and small I/Os, from different workloads, causing even longer latencies (I/O response times) for the smaller I/Os. Violin Memory recommends that you create separate LUNs for each workload (data, log and tempdb) to both isolate the workloads and allow for better parallelism through the operating system and file system.

3.2 Multiple Files and LUNs

With a traditional storage environment, file placement was critical to mitigate unbalanced I/O loads and other problems that could create higher latencies and possibly overload one or more of the disks, disk groups or arrays. With Violin Memory arrays, disk and RAID level issues go away so the number of files and placement is less critical.

However, there are considerations with SQL Server and the Windows NTFS file system that dictate having multiple files. The number of files and file placement really depends on the client's environment, but as a general rule the best practice recommendations for a new environment are:

1. One data file per physical core (not logical, hyper-thread core) with data access distributed evenly. This recommendation has relaxed over the years as NTFS has removed the file locking issue. Of more concern in higher performing systems is the data allocation over the many data files. If the data files are of varying sizes, data will not be allocated evenly and data access will likely be uneven as well.
2. The optimal number of tempdb files vary with the application. Violin Memory suggests starting with a number equal to ¼ to ½ the number of cores (between 4 and 8), and grow the number up to the total number of logical processors. The usage by SQL of the tempdb can be monitored with Performance Monitor or 3rd party tools. Having multiple LUNs is more important than having multiple files per LUN.
3. Windows allocates one I/O handing thread per underlying LUN (not Volume). This can create a bottleneck in both IOPs (I/Os per Second) and increase latency (I/O response time). Thus, it is highly recommended that you allocate multiple LUNs for both the data and tempdb workloads. Four LUNs is a good start for moderate performing databases with eight LUNs being optimal for the highest performance databases.

SQL Server's native backup process, for example, will create an I/O stream for every file, adding parallelism to the backup/restore that is negated by having few files.

3.3 Sizing Requirements

The old formula of sizing the required database space consisted of the actual size of data, index and logs, factoring in growth, size of each disk, latency of each disk, combined performance of a tray of disks, type of raid overhead and the desired performance. With flash-based arrays, this complicated process is no longer required as the low latency and high IOPs delivered by all-flash arrays mitigates the performance limitations of traditional arrays. Sizing files is now just concerned with the actual size of data, index and log space including growth.

3.4 Pre-Size database files; Disable auto-growth

It is Violin Memory's recommendation for stability of performance that all files should be pre-created to the maximum size that they will ever grow to and be configured to disable the automatic growth feature. This includes log and tempdb files.

While it may seem optimal to start the files small and let them grow to the size they need, this requires file growth operations, which pauses the database during the growth operation and can lead to uneven file sizes. Pausing the database can be harmful to applications and noticeable to users. It can be especially affecting if the growth amount is large and the operation takes many seconds to minutes. Uneven file sizes can lead to uneven data allocation and therefore uneven LUN access. Uneven LUN access can lead to reduced performance.

3.5 Thick, Thin and Data Reduction based LUNs

Modern enterprise level arrays have many features for optimizing performance, resiliency, cost and availability. Each feature comes with both optimizations and architectural considerations. For LUN types, Violin Memory offers thick, thin and dedup (data reduction including deduplication and compression) configurations. Each should be weighed for its value for each individual workload.

Thick LUNs are the fastest but also the least cost effective. These LUN types have the least amount of overhead associated with I/O servicing on the gateways and thus offer the highest IOPs (I/Os Per Second) and lowest latency (I/O response time). They also reserve all of the physical space requested so any unused space in the LUN, by the application, is not usable by any other process. These LUNs are recommended for workloads requiring the highest performance. Examples of these workloads would be the transaction log files and the tempdb files (both tempdb data and tempdb log). Both of these workloads are relatively small (gigabytes) compared to modern enterprise arrays (terabytes). Neither of these workloads will reduce in size by significant amounts. Both of these workloads are very influential to database performance.

Thin LUNs are very similar to their thick LUN counterparts and the recommendations and considerations are mostly the same. Thin LUNs do require a small increase in I/O service processing as they include additional pointers to the physical location of the data on the array.

The performance degradation is commonly negligible to the point that thin LUNs are recommended over thick LUNs in all but the most extreme latency (I/O response time) sensitive applications.

Data reduction-based LUNs are designed for a cost-effective storage solution. They require additional array-level processing to run the deduplication and/or compression processes. These additional processes, while saving space and thus cost, will also decrease the overall IOPs and increase the overall latency due to their increased processing requirements. Despite the additional processes, the technology being executed on an all-flash array means that the net performance is still many times better than a hard disk drive-based solution with or without data reduction technology. Data reduction LUNs are recommended for workloads in which cost effectiveness outweighs performance or deduplication values are very high. Examples of this would be email servers, SharePoint servers, file stores, VDI (virtualized desktops) or VSI (virtualized servers). For SQL Server databases it is recommended that you test data reduction based LUNs for the data files in comparison to the performance and cost reduction achieved by using the native compression inside of SQL Server hitting thin LUNs. In some circumstances, higher overall application performance and higher overall data reduction can be achieved by utilizing the native SQL Server compression technology as the database engine has the ability to look inside the data pages by knowing the table schema and data page metadata.

3.6 Columnstore Indexes

Columnstore Indexes are a new feature introduced in SQL Server 2012. This new index type utilizes a new I/O engine and thus can both move data much faster to and from storage and creates a different I/O pattern than standard data tables and indexes. It is recommended that you isolate these FILEGROUPS onto separate LUNs to offer optimal performance and to isolate the workload pattern away from the other database workloads.

3.7 In-Memory OLTP

While the feature's name implies activity entirely in the host server's DRAM, the changes being made to the data (inserts, updates, deletes), in most cases, still needs to be saved to persistent storage. As the In-Memory features increase the transactional performance of the applications, it also requires a higher performance (lower latency) storage subsystem to save the changes. The transaction log is still a transaction performance affecting process as the writes still need to commit to storage before the application can move forward.

Violin Memory recommends that you place the transaction log for each In-Memory database onto its own LUN to optimize performance for each database by not intermixing workloads. Also, like any other database, it is recommended that you separate the log and data files onto separate LUNs.

4 Benchmark Testing

SAN benchmarking is a common activity in both evaluation of new technology and verification of performance prior to deployment. As technologies change in the storage tier, so do changes in benchmarking in both tools and techniques. Below are some recommendations and considerations for benchmark testing with modern storage.

4.1 SQLIO & DiskSpd

SQLIO is one of the more common storage benchmarking tools utilized in Microsoft environments. It was developed many years ago when both hardware and software architectures were simpler. In reaction to this, Microsoft has release a new tool, DiskSpd, in which it now recommends as a replacement for SQLIO.

Some reasons for the new tool include NUMA awareness, advanced affinity settings, random file data generation for write testing against deduplication-based LUNs, better I/O and CPU-based reporting and mixed workload testing (no longer restricted to 100% read or 100% writes).

Violin Memory recommends using DiskSpd instead of SQLIO moving forward as it both allows for mixed workloads and allows for random data to be utilized inside the write I/Os, which can cause significant performance changes on data reduction (deduplication) based LUNs. IOMeter and vdbench are also recommended benchmarking tool options as they allow for these types of configurations as well.

4.2 Data reduction testing

As flash arrays deepen their penetration into the storage market, technologies like deduplication are becoming more common. Deduplication technology has been around for many years but the technology ends up randomizing the data pattern on the physical storage media and this is very harmful to hard disk drive technology's performance. Flash technology is not adversely affected by random I/O workloads so deduplication technology's cost reduction value can be rather appealing.

When benchmark testing storage appliances, new considerations must be made. Older benchmarking tools like SQLIO generate source testing files of a repeating pattern. All-spaces in the case of SQLIO. By writing I/O packets of all the same pattern the deduplication engine will reduce the physical I/O writes and hash table entries down to just one block. Some deduplication technologies even discard the I/O upon noticing a repeating pattern. While this is helpful in circumstances, like eager zeroing a virtual machine, this does not yield valid results for benchmark testing or applicable results for workloads like databases.

Violin Memory recommends using a benchmarking tool that allows for both a source file to be generated with randomized data (some tools, like vdbench, allow for the ratio to be specified) and to generate random data for the write portion of the workload.

Violin Memory also recommends using at least 8 LUNs when benchmarking with deduplication-based LUNs. This allows for optimal parallelization through all tiers of the I/O servicing process.

4.3 Number of LUNs

When benchmark testing storage subsystems, optimal performance is best achieved through utilizing multiple LUNs. This allows for parallelization through several layers including the operating system, file system, transportation system (fibre channel or iSCSI) and array. For optimal results Violin Memory recommends a minimum of 8 LUNs.

4.4 Number of threads & outstanding I/Os

There are two theories behind the configurations of benchmark testing. One theory is to test many combinations of threads and outstanding I/Os in order to eventually find the maximum performance results of the storage system. The other theory is to use one thread per logical core as that is what the database will do. In this theory, only outstanding I/Os is tunable. This theory is best applied to single, large database systems in which the database is the only process running on the server.

As long as each storage system is tested with the same general theory, benchmark testing will provide comparable results. It is most important to model the workload as best as possible to achieve the most real-world reproduction of application effect. What a SAN can do in general and what it will do for your application are sometimes two different things. Important factors to model would include read/write ratios, block sizes and application workload mixes (more than one workload hitting the SAN at the same time, like large data reads and small transaction log writes).

Also of note is the outstanding I/Os configuration. Up to a certain number, increasing this parameter will yield higher IOPs. Once a bottleneck has been hit (transport, OS threads, storage, etc.) then increasing this value will, at best, result in no performance changes and at worst artificially increase the latency (I/O response time) by stacking I/O requests in the transport queue. Make sure that the recorded latency is that of the test run just under the bottleneck as that is what the infrastructure is capable of.