WRITTEN BY Michael K. Campbell

TOP 10 WAYS TO INCREASE SQL SERVER PERFORMANCE WITH THE HARDWARE YOU ALREADY OWN

SUCCESSFUL SOLUTIONS NEED CONSTANT ATTENTION. Unlike their failed counterparts, successful solutions are successful solely because systems, users, and organizations have come to depend upon them. Not necessarily because they were well crafted, intelligently designed, or adhere to any other theoretical standards or best practices.

As a consultant, former DBA, and former database developer, I've seen far too many horribly designed, implemented, and managed solutions that were not initially successful, but still became mission-critical. The problem with these solutions – as well as with successful solutions that were intelligently planned – is when success draws too many users, applications, and business dependencies.

In essence, performance becomes a problem when a solution's success starts to overwhelm its bearing capacity. Some applications reach this point almost immediately – either because they were more successful than anticipated, or because they were horribly put together.

Consequently, truly successful applications are the kind that require attention because they're constantly changing and adapting to meet business needs—all while taking on greater load due to increased demands for the services they offer. As such, a key component of keeping successful solutions viable depends upon being able to stay ahead of performance requirements. And while one way to do that is to throw more and more hardware at problems as they crop up, another (better) option is to maximize existing hardware and resources to increase SQL Server performance with the hardware you already own.

About this White Paper

Nearly all successful applications can benefit from the performance-tuning techniques outlined in this document. However, the techniques outlined in this white paper should serve as a checklist for determining whether it might be time to consider throwing more hardware, or other resources (such as consultants, third-party solutions, and so on) at a problem.

Topics addressed in this white paper have been the focus of countless books, articles, white papers, and blog posts. As a result, this paper is not designed to provide an exhaustive overview, but to help raise awareness of potential performance-tuning techniques, provide background information and context for evaluating the effectiveness of these techniques, and to outline a couple of the key approaches and performance benefits that stem from implementing these approaches where applicable.

To that end, this paper provides a list of the Top 10 ways you can increase solution responsiveness, thereby increasing solution performance and success.

Top 10 Performance Improvements

Memory

While it's common for databases today to be over 100GBs in size, it's fairly uncommon to find a server with more than 100GB of RAM. More importantly, it's safe to say that a large number of successful databases today run on few GBs of RAM than the number of GBs required to store them on disk – meaning that the total amount of data to be served typically can't all fit in memory. If RAM were cheaper, then many performance problems would cease to exist – because SQL Server wouldn't have to rely as heavily upon disk (which is exponentially slower than RAM). Consequently, a good way to keep servers performant is to make sure you're using as much RAM as possible.

1. Enabling AWE Memory on 32-bit Systems

With 64-bit systems, enabling AWE Memory is not a concern. But with 32-bit systems, I'd wager that 50-60% of the SQL Servers running within Small to Medium Businesses (SMBs) are improperly configured – meaning that they're not using all available RAM.

In many cases as a consultant, I've been hired by clients to fix poorly performing queries only to find that SQL Servers with as much as 32GBs of physical RAM were only letting SQL Server use a paltry 2GBs (meaning that the remaining 30GBs was largely just sitting idle).

By way of background, one of the problems with x86 systems is that 32-bit pointers can only address, or reference, a total of 4GB of physical memory. Furthermore, on 32-bit systems, applications are restricted to using only 2GB of user-addressable address space (as the OS reserves the other 2GB for itself) . Microsoft's Physical Address Extension, or PAE, provides a workaround that forces the OS to work with 1GB of RAM, while giving applications up to 3GB of usable address space – but that hardly makes much of a difference on systems with over 4GBs of physical ram available.

For larger systems, Address Windows Extensions, or AWE, was created to 'extend' the window, or size, of addressable memory – clear up to 128GBs of RAM on 32-bit systems in the case of Windows Server 2003 Datacenter Edition. The only drawback to AWE memory is that it isn't dynamically managed through Windows' paging functionality like normal, virtual, memory. Consequently, applications requiring more than 2GB of RAM need to be granted the 'Lock pages in memory' User Right through the Local Security Policy.

Sadly, the installation wizard for 32-bit versions of SQL Server doesn't call awareness to this fact or present administrators with the option to configure AWE during setup. Moreover, even if Admins or DBAs grant SQL Server access to more than 2GBs of RAM using Enterprise Manager, SQL Server Management Studio, or T-SQL, those directives can't be honored by SQL Server until its service account has been granted the 'Lock pages in memory' user right. Large numbers of 32-bit SQL Servers run with access to only 2GBs of RAM when frequently there is much more physical memory available.

A quick way to 'spot check' for this problem in the wild on 32-bit systems is to add up the storage space of all actively used databases on disk, and then open up Windows Task Manager. If the system has more than 2GBs of physical RAM available, and the size of actively used DBs on disk is greater than 2GB in size, then you'll want to explicitly check for the 'Lock pages in memory' user right if you're only seeing 2GBs of RAM being used within Windows Task Manager.

If you do have this problem, it's quite easy to correct. You just need to grant your SQL Server's service account the 'Lock pages in memory' user right, reconfigure your SQL Server to use more RAM (if you haven't already done so), and restart SQL Server service so that it can take advantage of the additional RAM. I've provided step-by-step instructions for correcting this problem online , and remedying this problem will provide huge performance benefits with very little effort.

Disk
Disk, or the Input/Output (IO) sub-system plays a critical role in both database availability and performance. Therefore, ensuring that the IO sub-system is properly configured provides tremendous performance benefits. Unfortunately, correcting issues with the IO subsystem can be labor intensive as correcting poorly configured disks will require data on those disks to be migrated off such that the corrections can be made. In other words, data will need to be backed up, moved to another storage location, then disk configuration can be optimized, and data can then be restored or copied back into place. Consequently, these optimizations might not be feasible in some environments (where the requisite downtime may not be acceptable).But in environments where the 'luxury' of scheduling downtime is available, correcting these issues can substantially improve performance with no out-of-pocket costs.

2. Partition Offset Alignment.
With Windows Server 2008, Microsoft started to automatically address a common, and poorly documented, IO performance problem: Partition Offset Alignment. For systems running on older operating systems (Windows Server 2003 R2 and lower), handling this consideration needed to be done manually – which was rarely done since it was so poorly understood.

As Kevin Kline points out in his highly publicized article in SQL Server Magazine , an overly simplistic explanation of the Partition Offset Alignment problem is that before Windows Server 2008, the first block of data written to a partition was roughly 63k in size. Yet the OS likes to write 64k blocks of logical data to 64k sectors on the physical disk. Therefore, without explicitly aligning partitions, a single write of 64k of data will span two 64k sectors. The problem is then compounded as well for reads – and can impose a 30-40% performance penalty in many cases.

At a more technical level, partition offset alignment is simply a holdover from days when Windows Servers weren't equipped with RAID, and therefore didn't know how to align Operating System sectors with those of their underlying RAID configurations. Windows Server 2008 addresses this issue by automatically imposing an offset of 1024 – which should be enough for most systems. But this is only partitions that it defines (so be careful with upgrades).

Sadly, there's no real easy way to spot this problem in the wild on older systems. This will be something to look into if you're experiencing IO performance problems . This topic has received a lot of attention of late , and Microsoft has provided a fantastic white paper that covers this topic in depth – along with best-practices and solutions. The only drawback to correcting this problem, however, is that re-aligning partition offsets will remove all data on your disks – so this should only be done when you've got the downtime to move everything off your disks before making the change.

3. NTFS Unit Allocation Size

Conceptually similar to problems with partition offset alignment, NTFS Unit Allocation sizing can play a role in how efficiently Windows and the IO subsystem 'carve off' new chunks of disk when writing data. Windows will by default attempt to allocate new units in 4KB blocks – which is well suited for file servers with lots of small files. But since SQL Server reads and writes data in 8K pages and 64K extents, 4K allocation units typically end up being too small to be optimally effective.

Microsoft recommends a "64-KB allocation unit size for data, logs, and tempdb." However, it should be noted that a 64K allocation size may not be perfect for every system. In cases where scatter-gather IO (i.e. cases where SQL Server is executing seeks against small blocks of non-contiguous data) is more prevalent, 8KB pages may make more sense – especially if writes are much less common than reads. On the other hand, in cases where sequential IO is more prevalent, such as in reporting or analysis solutions, then larger allocation units (over 64K) may make more sense.

While tuning allocation unit sizes is easy (just specify the value you want from the Allocation Unit Size dropdown when formatting a disk), the problem is that it requires you to reformat your disks. So, this solution is best implemented when addressing one of the bigger changes listed above – and is really included here only for the sake of thoroughness – as this optimization won't have as dramatic of an impact on most systems as Tips 2 and 4.

4. RAID Levels

If you've addressed block-sizes and partition offsets, and are still running into IO performance problems, another key consideration is that not all forms of RAID are created equal.

RAID-0 is extremely dangerous in any production SQL Server environment and shouldn't be used. Using it to approximate production-level speeds in testing environments in some cases may make sense. Otherwise, steer-clear of RAID-0 entirely. And don't fall prey to the notion that you can use a RAID-0 for your tempdb, because a failure of a single disk in that array will take your entire server down.

RAID-1 provides basic fault tolerance by writing copies of data to more than one drive. Typically, write speeds on RAID-1 systems are the same as with writing to a single drive (theoretically there's some overhead at the controller level, but with most controllers these days that overhead is negligible at worst), whereas read speeds are potentially increased as data can be pulled from multiple drives when needed. RAID-5 provides excellent fault tolerance because it spreads writes out over the number of drives in the RAID (minus 1), and then writes parity data to the remaining drive in the array. This, in turn, means that the data on a lost drive can be reconstructed by extracting the data from the remaining drives and comparing it against the parity, or signature of the drive. As with RAID-1, RAID-5 systems pick up increased read performance because reads can be pulled from multiple spindles, or disks. Write performance, on the other hand, suffers a penalty because of the way that data has to be written to disk, scanned, and then written for parity or fault tolerance. Consequently, each logical write against a RAID-5 array incurs 4 physical IOs (2 writes and 2 reads). RAID-5 systems are therefore not recommended on systems where writes consistently account for more than 10% of overall activity. Backups and maintenance would obviously be exceptions here – IF you've got non-peak times in which these operations occur. To determine the percentage of writes to reads, you can simply use SQL Server Profiler and a bit of math . In my experience, only a small percentage of SQL Server environments incur greater than 10% writes on a regular basis – meaning that RAID-5 remains a solid and cost-effective solution in most environments.

RAID-10 overcomes the limitations associated with RAID-0 and RAID-1 by combining a high-degree of fault-tolerance with high-end performance – at the expense of drastically increased price. Consequently, if you need optimal disk performance, RAID-10 is commonly assumed to be the best path to take – when cost is not a factor. Be aware though, that RAID-10 is not a panacea, nor is it always, necessarily faster than RAID-5 .

Determining which RAID configuration to use is best done when setting up a system, because course-correction 'after the fact' is non-trivial in terms of effort and downtime. However, if you're experiencing IO performance problems , and have enough disks and suitable controllers available to deploy a new RAID configuration with existing hardware, then switching to a RAID-5 or RAID-10 solution can provide substantial performance benefits.

Otherwise, changing RAID types might better be considered either a last-ditch solution to improving performance with existing hardware, or an invitation to consider throwing newer and more expensive storage options at your problem – assuming that you've addressed partition offsets, block sizes, and are still having performance problems. Unless, of course, you're not using your existing disk resources as efficiently as possible.

5. Optimizing Disk Usage

Even with IO hardware perfectly configured, I find many organizations failing to make the best use of all available resources – especially when it comes to disk. First and foremost, since the IO subsystem is essential to both performance and availability, both considerations can be optimized when these different tasks are addressed separately.

For example, while keeping full backups on hand is a quintessential availability requirement, once a full backup has been made, it just sits around taking up space – unless it's needed. On the one hand, you can't risk availability or recoverability, even if dedicating high-performance disk-space to backups means that you have less disk to dedicate to indexes or other performance needs. I commonly recommend that organizations keep one or two days' worth of backups on-server and then push copies of older backups out to a file-server, network-share, or other less high-performing (and less expensive) storage mechanism.

By automating the process of moving backups to another location, it's possible to free up more expensive and better performing disk locally – such that it can be better used for more performance oriented endeavors. This is also a case where third-party backup solutions make sense too – as they can decrease the amount of disk needed to store backups, thereby allowing more of your 'premium' disk to be used for log, data, and index files.

Another common scenario within many organizations is a tendency to store all SQL Server log and data files on the same volumes – even when other volumes are available. In these cases, log and data files compete for IO on the same set of spindles, while other spindles (or volumes) effectively remain idle. My guess is that this approach to storing SQL Server data and log files is done to keep things more organized and easier to manage – but such an organized approach to the universe comes at the cost of decreased performance. As such, if you want to keep things organized across volumes, I recommend creating a SQLData or SQLLogs folder on each volume – as that can help make it easier to keep SQL Server data and files contained, while still spreading it out over multiple volumes. Note that splitting index files out over additional volumes is a huge way to boost performance. You won't get this same performance benefit when splitting SQL Server data files over different partitions on the same volume – only when you're spreading your load out over additional spindles, or volumes. But by splitting data, log, and index files over distinct drives, you can typically boost performance in impressive ways.

So, when it comes to optimizing disk performance – make sure that you're intelligently balancing your workload over as many (redundant) disks as you have available. And, try to offload logging and data storage to your fastest volumes while relegating backups and other needs to less-performant disk whenever you have the option.

Network

Most SQL Servers see a fair amount of traffic from applications and users. However, with the exception of backups, it's uncommon for most SQL Server deployments to saturate network connections. But that doesn't mean that minor modifications and network configuration changes to existing hardware can't yield big performance improvements.

6. Jumbo Frames
Standard Ethernet frames are limited to 1,500 bytes per payload – which require overhead to manage when sending out large streams of data since that data has to be 'chunked' into small payloads. Jumbo Frames overcome this issue by boosting the amount of data per payload up to 9,000 bytes – thereby reducing the processing and management overhead needed to send data back and forth across the wire.

Setting up jumbo frames on a Gigabit (or better) network can be tricky, and it's not uncommon to run into problems while attempting configuration changes. You'll want to schedule downtime and plan and test accordingly before making this change in production environments . Jumbo Frames may be one way to increase overall performance and responsiveness if you're constantly sending larger blocks of data from 64-bit SQL Servers to 64-bit application servers (for reporting or other, similar, purposes).,

Maintenance
There's no denying the benefit that a full-time DBA can impart when it comes to managing performance considerations and needs – especially through finely-tuned and well-regimented maintenance plans. But in organizations where a DBA isn't warranted (such as when databases are small), or available, the automation of regular maintenance tasks can provide tremendous performance benefits.

7. Updating Statistics
One of the things that make SQL Server so successful is its phenomenal internal query execution engine – which relies on complex statistics about the nature, cardinality, and density of data to speed the execution of queries. However, without regularly updated statistics, SQL Server's ability to properly determine the most efficient (or one of the most efficient) approaches to satisfying a query starts to wane. SQL Server is by default set to update statistics automatically, but in my experience it never does a good enough job of staying on top of updating statistics on its own.

Moreover, in systems where large volumes of changes happen regularly (or even occasionally), it's easy for statistics to get out of whack, and for performance to suffer. Even worse, if statistics are allowed to degrade over time, it's not uncommon to encounter situations where the slow buildup of statistical 'sludge' will lead to performance problems that slowly degrade until reaching a 'breaking point' – where performance can take a catastrophic turn for the worse with some key queries or operations.

While an in-depth overview of updating statistics is outside the scope of this document, an easy way to achieve roughly 70% of the benefit of a full-blown statistics updating routine with about 2% of the effort is to simply schedule the following query as part of a SQL Server Agent Job that runs every night during non-peak times:

```
EXEC sp_msForEachdb
       @command1 = 'EXEC ?..sp_updatestats'
```

While not perfect, this simple script will go through each database on your server (using the undocumented sp_msForEachdb special stored procedure), and do a bare-minimum update of any statistics that have gotten wildly out of whack.

Additional resources and insights into routinely updating statistics can be found in Books Online , and through several articles and blog posts online . The key thing to remember is that without regularly updating statistics, performance will suffer – even if SQL Server is configured to automatically update them.

8. Defragmenting Existing Indexes
Another regular task that DBAs undertake is to routinely defragment indexes. If you're not familiar with index fragmentation, an overly simplistic way to think of it is to relate it to disk fragmentation – where writing data causes it to be placed haphazardly so that reading this

randomly distributed data incurs more IO operations than what would be optimal. Only with index fragmentation, things are more complex because fragmentation can stem from data in index pages being out of logical order, or from data within index pages being too thinly populated. In either case, performance degradation occurs and becomes worse and worse as the problem goes uncorrected. In fact, in a white paper detailing index fragmentation for SQL Server 2000 , performance penalties for fragmented indexes ranged from 13%-460% depending upon environment .

A thorough overview of how to defragment indexes is outside this paper's scope, but, Michelle Ufford provides a top-notch Index Defrag Script that can be easily implemented and automated in environments without a full-time DBA. (I personally maintain that you can't be a decent DBA until you've written at least two index defragmentation automation scripts, and even though I've written more than enough of my own, I really like Michelle's because it's very thorough yet remains very simple and easy to use.)

Advanced Indexing Tasks
Pound for pound, indexes are more efficient that hardware because properly defined indexes reduce the amount of load that needs to be placed on hardware. Properly configuring indexes, on the other hand, requires a thorough understanding of T-SQL, SQL Server internals, and indexing techniques and best practices. As I like to say, index tuning is 50% science, 50% skill, and 50% dark magic. But properly managing indexes can typically provide the most amount of performance improvement with the least amount of cost.

9. Removing Unused Indexes
While properly defined indexes will drastically improve overall performance, it's not uncommon for many organizations (especially those without a full-time DBA) to assume that the answer to all of their performance problems is to add more and more indexes. However, every index added to a database comes with a price – in the sense that the index has to be updated each and every time there is a change (UPDATE, DELETE, or INSERT). In most cases, the performance benefits of indexes far outweigh the overhead associated with maintaining indexes (especially if indexes are regularly defragmented).

In cases where indexes are infrequently or never used to satisfy a query, they only add overhead during updates or modifications. As a result, unused indexes detract from performance instead of improving it – and should be removed.

While removing unused indexes is effectively a trivial task that can be done either through the UI or by using the DROP INDEX T-SQL command, finding those unused indexes can be a complex affair. You should first start by looking for indexes with poor selectivity – where selectivity is a term used to define how unique (or capable of selecting, i.e. selective) the values are within an index. While removing unused indexes is effectively a trivial task that can be done either through the UI or by using the DROP INDEX T-SQL command, finding those unused indexes can be a complex affair. You should first start by looking for indexes with poor selectivity – where selectivity is a term used to define how unique (or capable of selecting, i.e. selective) the values are within an index.

For example, in a company with 1,000 employees, and index on Date of Birth would likely be highly selective – meaning that a query for employees born on a given day should, statistically, never return more than an average of 3 or 4 employees per day. In SQL Server terminology, this index would have a density of .003% (or 3 out of 1,000) - which, in turn, would translate to an index selectivity of .997 (index density and selectivity are inversely related or proportional.) Essentially, this index would be beneficial in any query used against the birth date of employees – as the data within this index is 'selective' or capable of discriminating against different types of results.

Within SQL Server, the more selective an index, the greater the chance that it will get used – and the more efficient it will be at returning results in a performant manner. For example, imagine another index – this time on the EmailAddress column of the same employees table. In this case, each entry has to be unique (assuming that employees can't be listed more than once), meaning that the density of this index would be .001 (or 1 out of 1,000) – giving it perfect selectivity.

One the other hand, an index on the Gender column of the same table would be a very poor index because it only has two unique values, and (in most cases) would have a fairly even distribution of values throughout the table – making the index largely useless when it comes to querying data. In fact, as a rule of thumb, unless an index is a covering index (covered in the next tip), SQL Server won't use an index with a selectivity of less than 90% - meaning that it should be removed.

To calculate index density and, thereby, specificity, use the DBCC SHOW_STATISTICS command against a target index on a given table. For example, to check on one of the indexes mentioned previously, you should run the following:

```
DBCC SHOW_STATISTICS(Employees, IX_Employees_ByGender)
```

Then, in the first set of results, check the Density column – along with the "All density" columns from the second-set of results. Remember that these values are in exponential notation (so 1.166079E-06 is actually 0.00001166709) whenever there's an E-0x in the results. Otherwise, as long as the density value returned is less than .1 (i.e. a selectivity of more than 90%), it's a pretty safe assumption that the index is selective enough to be used in most cases.

Of course, it's also possible that some highly selective indexes exist, but never get used because no queries are ever fired against the columns being indexed. Cases like this are much harder to find, but where present still leech performance overhead during modifications. While a number of techniques to find these kinds of indexes exist , removing unused indexes can provide a huge performance boost in environments where indexing has gotten out of hand. (In cases where the addition of indexes has been more controlled and deliberate, you're not typically going to see huge results – unless you've got a specifically large index that sees lots of updates.)

10. Creating and Tuning Indexes

Sadly, creating and optimizing indexes is outside the scope of this paper. However, the proper configuration and use of indexes is one of the easiest ways to dramatically increase SQL Server performance – with virtually no out-of-pocket costs. So, if you're unfamiliar with the basics of creating indexes, a review of some online overviews of indexing basics can be highly beneficial .

However, index creation and tuning is hardly something that can be mastered by reading even a large number of articles or even books. Instead, learning how to properly tune and create indexes will take a good deal of research along with some hands-on experience.

Even the basics of index tuning can pay massive dividends when it comes to increasing SQL Server performance. More importantly, what makes indexes so powerful is that when correctly employed, they can drastically cut down the amount of data that SQL Server needs to examine when completing a query – even when millions of rows are involved. This, in turn means that by effectively using indexes, you can drastically reduce the demands placed upon your hardware – meaning that you can do more with the hardware you already have on hand.

ABOUT THE AUTHOR Michael K. Campbell is a SQL Server consultant, author and founder of SQLServerVideos.com, home of high quality, FREE SQL Server instructional videos

Idera | BBS Technologies | 802 Lovett Blvd Houston, Texas 77006
PHONE 713.523.4433 | TOLL FREE 877.GO.IDERA (464.3372)
FAX 713.862.5210 | WEB www.idera.com